2003-03

An XML-based knowledge management system of port information for U.S. Coast Guard Cutters

Stewart, Jeffrey D.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/1062

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN XML-BASED KNOWLEDGE MANAGEMENT SYSTEM OF PORT INFORMATION FOR U.S. COAST GUARD CUTTERS

by

Jeffrey D. Stewart

March 2003

Thesis Advisor: Magdi N. Kamel
Second Reader: Gordon H. Bradley

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: An XML-Based Knowledge Management System of Port Information for U.S. Coast Guard Cutters	**5. FUNDING NUMBERS**
6. AUTHOR(S) Stewart, Jeffrey D.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis describes the development of a prototype application which collects, manages, and distributes knowledge gained by Coast Guard cutter crews making port calls throughout the world. The system uses XML technologies in server/client and stand alone environments. With a web browser, the user views and navigates the system's content from a downloaded file collection or from a centralized data source via a network connection. Users add and modify content with Hypertext Markup Language (HTML) forms using their existing network connections. Client-side data access and navigation, as well as data storage, is performed using non-proprietary standards developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

The prototype application's purpose is to fulfill the strategic goal of achieving superiority of maritime domain awareness over the areas in which the Coast Guard operates. The need for this application is based upon the lack of specific information from currently available reference publications, the absence of a system to distribute port call knowledge, and the data bandwidth limitations of cutters at sea. The need for knowledge retention aboard cutters is elevated by shortened crewmember assignment lengths due to the stressful and arduous duties of life at sea.

14. SUBJECT TERMS Knowledge, Knowledge Management, XML Database, Extensible, XML, XPATH, XSLT, XML Schema, XQuery, XLink, XPointer, XInclude, CSS, HTML, XHTML, DOM, JavaScript, ASP, Internet Explorer, MSXML, Coast Guard, USCG, Cutters, Personnel, Navigation, Ports, Locations, Experiences.	**15. NUMBER OF PAGES** 123
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AN XML-BASED KNOWLEDGE MANAGEMENT SYSTEM OF PORT INFORMATION FOR U.S. COAST GUARD CUTTERS

Jeffrey D. Stewart
Lieutenant, U.S. Coast Guard
B.S., U.S. Coast Guard Academy, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author: J. D. Stewart

Approved by: M. N. Kamel
Thesis Advisor

G. H. Bradley
Second Reader

D. C. Boger
Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis describes the development of a prototype application which collects, manages, and distributes knowledge gained by Coast Guard cutter crews making port calls throughout the world. The system uses XML technologies in server/client and stand alone environments. With a web browser, the user views and navigates the system's content from a downloaded file collection or from a centralized data source via a network connection. Users add and modify content with Hypertext Markup Language (HTML) forms using their existing network connections. Client-side data access and navigation, as well as data storage, is performed using non-proprietary standards developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

The prototype application's purpose is to fulfill the strategic goal of achieving superiority of maritime domain awareness over the areas in which the Coast Guard operates. The need for this application is based upon the lack of specific information from currently available reference publications, the absence of a system to distribute port call knowledge, and the data bandwidth limitations of cutters at sea. The need for knowledge retention aboard cutters is elevated by shortened crewmember assignment lengths due to the stressful and arduous duties of life at sea.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I could not have finished this without the support of my wife, Kristin.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. RESEARCH CONDUCTED

The purpose of this thesis was to construct a server based XML knowledge management system that can be used in a network-less environment with Internet browser software. The users of this system are United States Coast Guard cutter personnel. The system captures their corporate knowledge and experiences while visiting ports of call and navigating their geographical areas of responsibility. While built for a specific knowledge domain and user group, the methodology of this research can be extended to any professional community that works in low or no bandwidth environments where knowledge is critical to the outcome of their performance.

1. Research Questions

The primary research question was: how do we build collaborative extensible markup language (XML) based knowledge management systems that are transparent, deployable, and extensible using ubiquitous technologies and accepted standards? The terms used in the primary question are given context with the following definitions:

- **Collaborative** – anyone can author and comment on database content.
- **Knowledge** – the applied information, experiences, and tacit rules.
- **Transparent** – accessible without middleware (i.e. direct access to data).
- **Deployable** – the database can be packaged and used from the client software without a server or Intranet/Internet connection.
- **Extensible** – existing content can be expanded, and new bodies of knowledge can be created using the same concept and construct.
- **Ubiquitous technologies** – Internet Explorer, HTML, JavaScript, and World Wide Web Consortium (W3C) languages.
- **Accepted standards** – W3C Recommendations and Internet Engineering Task Force (IETF) standards.

Several secondary questions are germane to this topic: What are the advantages and disadvantages of using XML over relational and object-relational database systems? What are the differences between currently available open-source and commercial XML database systems? What do emerging XML technologies such as XPointer, XLink, and

XQuery hold for the future acceptance of XML database applications? These topics were considered during the construction of the application and were further analyzed after completing the development process.

2. Outcome of Application Development

Research into the problem space began during the formulation of the thesis proposal. At that point, several principles of hypertext linking and XML processing were investigated for their potential use in a database consisting of a hierarchy of disk directories and files. The research then proceeded with two distinct iterations. The first iteration was a proof of concept – it was largely an exercise to prove that the envisioned system could be constructed and to increase the knowledge and skills of the author. In the second iteration a prototype application was constructed. The developed application demonstrates that a deployable XML-based knowledge management system is possible. Further research is needed to determine requirements and to validate the scalability and reliability of such a system.

B. THE NEED FOR AN APPLICATION

1. Knowledge Management

Admiral Collins, Commandant of the Coast Guard, outlines three areas of focus in his direction statement, one of them is readiness. To improve the service's current and future readiness, his direction is to, "[d]esign and implement a maritime domain awareness capability that provides integrated afloat, ashore, and airborne C4ISR that is focused on meeting both the informational needs of decision makers and the tactical needs of operational commanders (Commandant's Direction, 2002)." This thesis is about building the Coast Guard's ability to capture, store, and share the knowledge gained by its men and women serving aboard its cutters. It is focused on retaining the knowledge lost in the cycling of personnel rotations. The imperative to develop a solution to this problem is further discussed in Chapter II.

2. Disconnected Environments

People in the telecommunications profession often talk about the difficulty in getting high data bandwidth "to the last mile." For ships at sea, the last mile can often be more than a thousand miles. To connect these resources, radio-based communications

networks have been created to link to these resources. However, even the best of long-range radio networks (i.e., satellites) are slow in comparison to land-based networks. They are also very expensive to build and use. As such, their use is prioritized to operationally critical tasks.

The bandwidth limitation for ships at sea is not likely to be completely removed in the near future. This creates an environment where moored ships have high speed connectivity from a pier, only to be severely restricted while underway. Given this situation, there are opportunities to maximize the power of moored connectivity to lessen the impact of being nearly disconnected away from the pier. The developed application is one solution. It creates a collaborative environment for moored ships to share their newly gained experiences so that it may be captured as a 'snapshot' and used by other ships about to get underway.

C. SCOPE OF RESEARCH

1. Prerequisite and Learned Skills

This research covered many fields of computing: data theory and database design, networking fundamentals, software analysis and design, and client browser and web server processes. The author has attended courses in all of these areas, including introductory and advanced XML courses. In addition to putting that knowledge to practical use, the author invested significant learning time into the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), Document Object Model (DOM), JavaScript (ECMAScript), Visual Basic Script (VBScript) and numerous Extensible Markup Language (XML) technologies (XML, XPath, XSLT, XML Schema, etc.). Having solved many issues regarding a specific path of development, the necessary training time to re-create the skill set is estimated to be three months.

2. Program Management

This research was primarily focused on the technical solution, and less on satisfying the requirements of the user domain. As a proxy, the author used his own six years of afloat experience to create a requirements document. Other supporting documents used in defining the requirements are described in Chapter II. The time spent solely on this research is estimated to be nine weeks. Of that, approximately one-third of

the time was spent formulating a development path (i.e. trail blazing). Applying the same process into a separate community of knowledge could be repeated in approximately five weeks.

3. Computing Resources

The development tools and resources used were openly available to all students attending the Naval Postgraduate School. The primary tools and resources are listed below:

- Microsoft Visual Studio .Net – used for authoring scripting and CSS files. The VS.Net IDE provides significant value with its dynamically generated contextual help references.
- Altova XML Spy (Professional and Personal version 5) – used for authoring XML, XSLT, and XML Schema documents. This product was found to be a superior product to all others evaluated.
- Microsoft FrontPage (version 2000) – used for developing HTML templates. This product was also used for rapid prototyping of HTML layouts because of its fast rendering WYSIWYG interface.
- Server computer - Windows 2000 Advanced Server (service pack 3), Internet Information Server (IIS version 5.0), and Microsoft XML Parser (MSXML version 4.0, service pack 1).
- Client computers - Windows operating system (2000 Professional and XP Home), Internet Explorer (version 5.5 with MSXML 3.0 and version 6.0 with MSXML 4.0). Since NPS client computers do not have MSXML version 4.0 installed, client-side user data validation could only be tested from the author's personal computer.

D. THESIS OUTLINE

Chapter II is the requirements analysis of the developed application. It discusses the context of the user domain and provides the needs analysis for the Coast Guard. Without prescribing the solution, the chapter also describes the design criteria. Chapter III provides an introduction to the technologies used in researching and developing the solution. More detailed issues regarding these technologies are described in the later chapters. Chapter IV is a description of the proof of concept application. It demonstrates a less than optimal solution, but resolves many of the challenges to the development process. Chapter V is the second iteration of development and produces a superior solution compared to the first. It continues the process of identifying and solving development problems. Chapter VI provides perspective on the XML database products

that are currently available and introduces advanced XML technologies that will influence the future of XML databases. Finally, Chapter VII provides reflection on the application developed, its potential future, and the need for further research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. APPLICATION ANALYSIS AND DESIGN

A. INTRODUCTION

The purpose of this chapter is to provide relevant context for the prototype application. It is a requirements document for a particular instance of a native XML knowledge management system. This chapter communicates the analysis and design for a particular problem area; the following chapter describes the actual technologies used in developing the application (solution).

The requirements are documented using the Unified Process (UP) and the Unified Modeling Language (UML)[18, 48]. The actual writing of the documentation came after constructing the proof of concept application. This was intentional. It was necessary since the course of development took several major deviations along the way. The analysis and design documentation will support anyone else pursuing further study in this topic area or for an enterprise implementation of the application. The author chose the Unified Process to continue honing his skill in this technique and because it is suitable for an iterative development approach. This chapter focuses on those artifacts of the Unified Process that would likely be produced during the Inception phase of development. Chapters IV and V cover the later phases of Elaboration and Construction as specific iterations.

B. VISION

The short-term rotation of personnel aboard Coast Guard cutters inhibits the organization to achieve superiority in Maritime Domain Awareness (MDA). When personnel report to a new assignment (new by geography or type of mission) they begin learning about their area of responsibility (AOR), available resources, and points of contact within their work domain. The individual's expertise reaches a peak just before they are transferred to a new assignment. There is currently no effective means for capturing the corporate knowledge of Coast Guard employees. This affects unit performance and readiness, effectiveness with local 'business-partners', and future planning decisions. A successful solution would be a systematic collection of experiences, lessons learned, and recommendations from the current experts that would

be instantly available for study upon arrival at a new assignment, and be continuously refined as the region changes and new experiences occur.

For personnel assigned to Coast Guard Cutters, the knowledge about ports, resources, and people is invaluable. The at-sea portion of the Coast Guard's work is at the core of its service to the Nation. A collaborative knowledge system in this area may provide significant value added to the organization's performance of its missions.

C. DOMAIN MODEL

The purpose of the Domain Model is, "a representation of real-world conceptual classes, not of software components."[18] Research into conceptual classes began with analysis of similar objects existing within the seagoing communities of the United States Navy and Coast Guard. Both organizations have used the Logistics Requests (LOGREQ) system for years. The data fields within a LOGREQ and their explanation come from the U.S. Navy's Naval Warfare Publication (NWP) 1-03.1 (old 10-1-10). In addition, Navy ships also report their activities and discoveries through messaging and formatted letters as required by the Navy Lessons Learned System (NLLS), Naval Operations Instruction (OPNAVINST) 3500.37(series). Both documents, and the author's experience were used to evaluate, organize, and subsequently elaborate the detailed attributes of the conceptual classes.

Numerous information sources already exist for sailors. Commercial for public (e.g., Lloyds), government for government, and government for public publications provide generic information about locations, navigation, and ship related resources. From empirical use, these sources provide good information, but are insufficient to meet the Coast Guard's internal needs. The prototype system will supplement the areas where existing knowledge is lacking. However, the published documents provide a baseline for organizing and classifying the ports of the World. The World Port Index (WPI)[30] (a National Imagery and Mapping Agency (NIMA) publication), and the Coast Pilot[29] (a National Oceanic and Atmospheric Agency (NOAA) publication) were used in organizing the logical layout of the system. NIMA provides a schema for sub-dividing the world, which the WPI uses as a baseline for further categorizing its information. At the national level, the Coast Pilot organizes maritime locations within the United States

(and its territories and possessions). Both sources of information were used in developing the organization of the system. Foreign locations are further described in more detail by the Sailing Guides, which are published by NIMA. In addition, the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) has their own unique system for organizing world locations [24]. Using pre-existing models as a development template helps ensure a familiar data set for the user to navigate.

1. Domain Model Conceptual Classes

The following list contains the conceptual classes for the application developed:

- **Locations** - geographical points or distinct areas (e.g., New York Harbor, Antarctica).
- **Routes** – navigational tracklines for navigating to the Locations.
- **Piers** - permanent man-made structures to moor ships.
- **Anchorages** - areas within a location to moor an anchor to the water's bottom.
- **Resources** - business entities useful in fulfilling a ship's logistical needs (e.g., grocery, repair, etc.).
- **Activities** - entities and areas for having fun (for the morale and welfare of a ship's crew) during a port call to a specific Location. Both Resources and Activities may be businesses, but they differ in how and whom they serve (e.g. bowling alley versus a plumbing supply shop).
- **People** - persons associated with the Location, or to Resources or Activities within the Location.
- **Media** - visual and audio files stored and made available for users to view and download.
- **Comments** - chunks of additive information associated with the above-mentioned conceptual classes. Comments are designed to further clarify or explain existing content, without overwriting the original source of information.

2. Domain Model

Figure 1 is the domain model for the constructed application. Lines connecting the conceptual classes together indicate their associations. *Locations* form a relative hierarchy as parent and child relationships of each other. For a specific *Location*, there may exist content related only to that *location*. For each type of content, there may exist *Comments* about a specific content node. *People* may be connected to the *Resources* they work for or *Activities* they represent.

The multiplicity between classes is also indicated on the lines connecting them ('0..1' meaning optional but not more than 1, and '*' meaning infinite). For example, a *Location* may have zero to many sub-*Locations*, and zero to many content types. Each content type may have zero to many *Comments* about a specific content node. A *Person* (within the *People* content type) may be associated with zero or one *Activity* or one *Resource*.

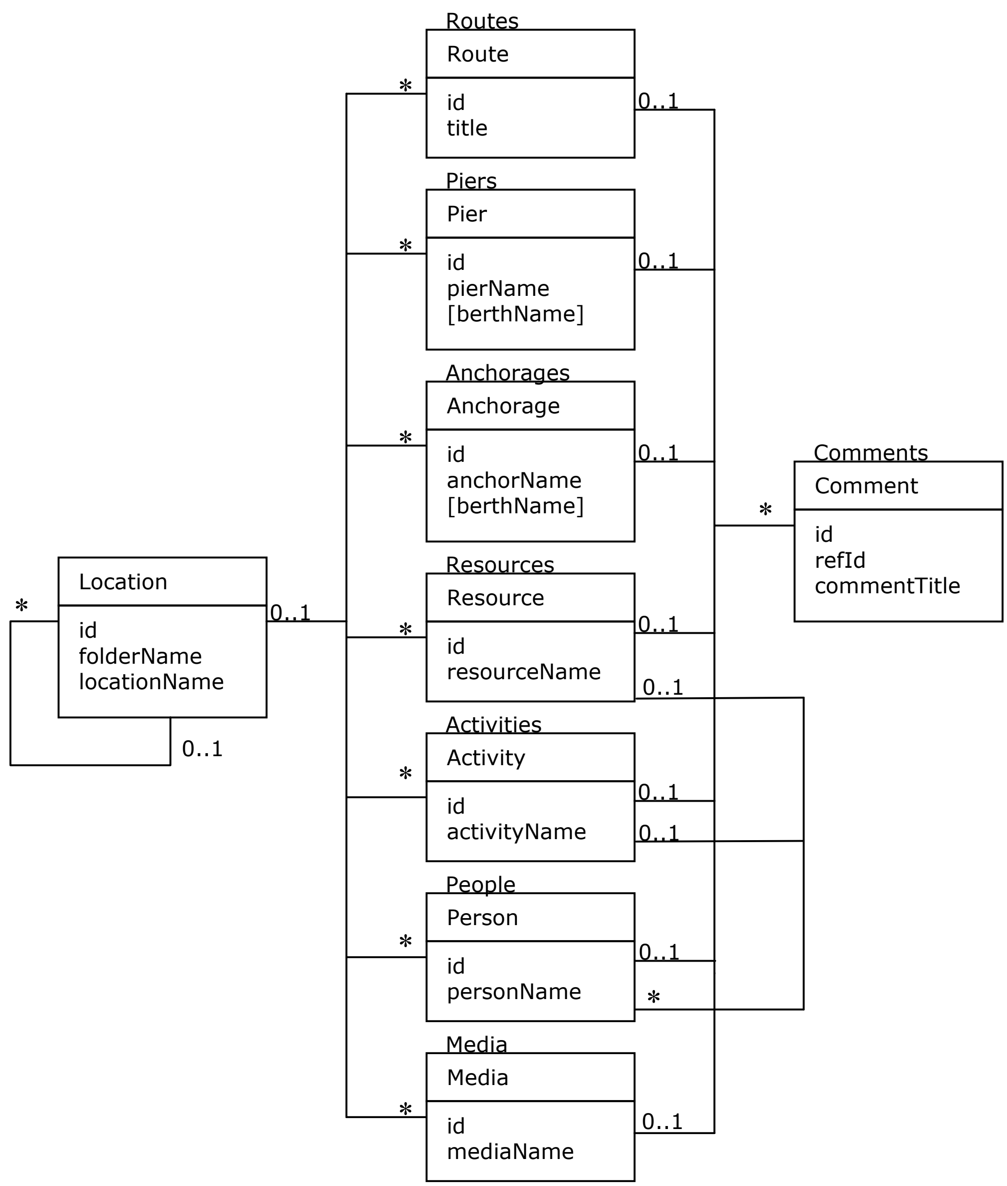


Figure 1. Application Domain Model

3. Key Attributes

The key attributes for each conceptual class are listed in Figure 1. Each content type is identified by a unique name and each content node is identified by a unique identification (id) value. A global index shall contain basic information and the *id* value of all *Locations*. Each *Location* shall contain basic information about which types of content have been created for that *Location*.

While not indicated, each content node shall also contain information about the individual who authored the content, how to contact that person, when it was authored, and also retain the average vote score based upon other users opinions on the value of the content provided (or modified).

Detailed attributes for each content type will be discussed in Chapter V. Use of key fields and their limitations on the performance of the developed application is also discussed in Chapter V. Examples (as templates) are contained in Appendix 2.

D. USE-CASE MODEL

1. System Boundary

For the application being developed, concepts within the system boundary include:

- The server(s), which hold the centralized content of the system, and its web hosting and XML processing components. The server also contains the application and its associated content (stored either locally or in a distributed manner).
- The client computers with which the users interact with the system, and its agent software (browser) and XML processing components.
- The organization's corporate-wide Intranet.

2. Actors and Goals

Those entities that are outside the system boundary but interact with the system are illustrated in Figure 2 and summarized below.

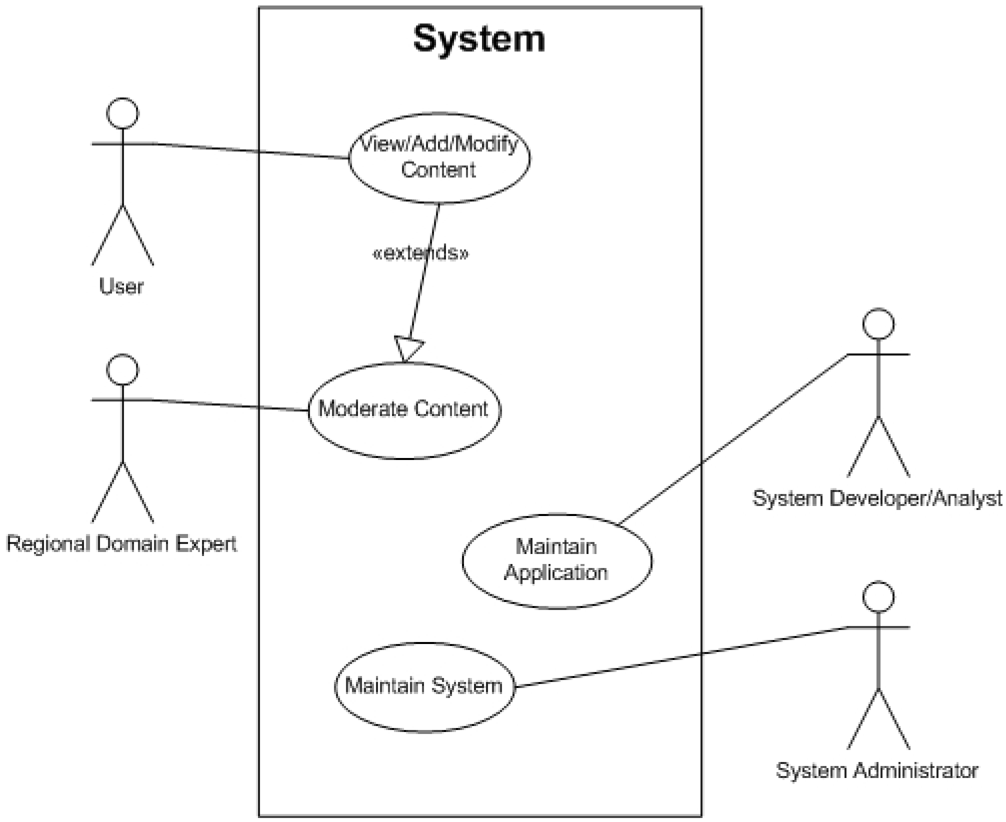


Figure 2. System Use Case Diagram

- Users - search and view content, add/modify (author) content, add comments about existing content, provide evaluation of existing content.
- Regional domain experts - moderate content (summarize, revise, correct), delete inappropriate or value-less content.
- System developers/analysts - maintain existing processing application, extend/deprecate content, create new tools, and maintain structure of the content.
- System administrators - maintain access accounts for regional domain experts and system developers/analysts, maintain server hardware, maintain operating system, and maintain web server and XML tools.

There are no actors external to the system that are systems themselves (excluding client browsers). The author acknowledges that this exclusion is intentional for the purpose of simplification.

3. Use Cases

For the actors described above, the following use cases are provided using an essential style (i.e. user-interface independent) in a casual manner. Elaboration of the actor's interaction with the system is described in more detail in chapters IV and V.

a. Users

(1) Search and View Content. Main success scenario: a user navigates through the hierarchical levels of locations within the system or searches on location key words to find a location in the database. Once found, the user reads and uses the available content.

(2) Add/Modify (Author) Content. Main success scenario: having experienced traveling to a location, the author logs onto the system and adds new information to system. Alternate scenario: if the already content exists in the system and no significant modification to the data needs to be made, the author may add a comment rather than modifying the existing information.

(3) Add Comments About Existing Content. Main success scenario: a user desiring to amplify or clarify existing content may log onto the system and then add comments to any content, without modifying existing content.

(4) Provide Evaluation of Existing Content. Main success scenario: a user may 'vote' on any existing content as to its overall quality and value to other users. The votes are anonymous. Votes provide a feedback mechanism to the system users, authors, and regional domain experts, without making formal comments about the content.

b. Regional Domain Experts

(1) Moderate Content (Summarize, Revise, Correct). Main success scenario: with the same functionality as modifying content above, the domain expert shall be able to log onto the system and then cull through existing content and cumulative comments to 'refresh' the content. The difference between user modifications and domain expert modifications is the clearing affect on comments. That is, comments remain

associated with content that is changed by users, but are deleted when domain experts 'refresh' the content.

(2) Delete Inappropriate, Inaccurate, or Low-Value Content. Main success scenario: the domain expert can remove content from a particular location (i.e. delete) when he/she determines it inappropriate, inaccurate, or of low-value.

c. System Developers/Analysts

(1) Maintain Existing Processing Application. Main success scenario: system developers/analysts shall have sufficient access to maintain and repair the system so data views and entry remain functional.

(2) Extend/Deprecate Contents. Main success scenario - system developers/analysts shall have sufficient access to create new content types and deprecate outdated content type or versions of content structure.

(3) Create New Tools. Main success scenario: system developers/analysts shall have sufficient access to implement new tools and capabilities to better meet user needs and harness the progression of the implementing technologies.

(4) Maintain Structure of the Content. Main success scenario: system developers/analysts shall have sufficient access to manipulate the hierarchy system and associated indexes to maintain application functionality.

d. System Administrators

(1) Maintain Access Accounts for Regional Domain Experts and System Developers/Analysts. Main success scenario: the system administrator logs onto the system and adds/modifies/deletes user accounts. Note that actual implementation of this is beyond the scope of this thesis and in actuality would involve the use of an actor that may be from an authentication service.

(2) Maintain Server Hardware. Main success scenario: the system administrator prepares routine backups of the system's contents, replaces inoperative hardware, and scales the capacity of the system as it grows in size and usage.

(3) Maintain Operating System. Main success scenario: The system administrator logs onto the server’s operating system and completes routine maintenance to the system as dictated by standard practices.

(4) Maintain Web Server and XML Tools. Main success scenario: The system administrator logs onto the server's operating system and loads updates to installed XML processing APIs and other internet service software.

E. SUPPLEMENTARY SPECIFICATION

The following briefly states the supplementary specifications for the system being considered.

1. Functionality

- The systems data must be viewable using a client browser without the requirement of a network connection. That is, using a de-compressed file (e.g. from a previously downloaded .zip file or CD-ROM disk) from the networked application, all navigation and view portions of the system must work correctly.
- Content node *ids* shall be a date-time stamp (a integer value) when the node was created. Subsequent modifications to the node shall retain the same *id* until modified by a domain expert (at which time the node shall receive a new *id*).
- User logon is intended to capture point-of-contact information, and not for security information. Therefore, less exacting identification methods and flexible contact data capturing methods shall be implemented.
- User accountability shall be traceable by means of Internet Protocol (IP) address capturing and time stamping. Data gathered at time of authoring content provide a traceable path back to the source on a case-by-case basis.
- Content modified by users shall not be deleted, but retained in a collection of modified content nodes within the same area as the corresponding location.
- Media files available to be uploaded into the system shall be limited to those media types that are supported enterprise-wide and shall also be restricted in size to approximately 1/20,000th of the systems total available disk storage space (e.g. an 80GB hard drive equates to a maximum media file size of 4.3MB each).

2. Usability

- User interface design shall be kept consistent throughout the system. Navigation toolbars shall be positioned similarly on every view page. Help and example information shall appear in the area (as a pop-up box) to which the form field refers.
- Hyperlinks and function calls shall behave in similar form of colors and mouse-over effects.

3. **Reliability**

- The implemented server shall maintain availability at 98% of the time, with downtimes not to exceed 2 hours on average. The intent of the system is to function as a repository of knowledge, and thus has no extraordinary mission-critical reliability requirements.
- Unanticipated downtimes exceeding 30 minutes and other maintenance outages shall be reported to accessing users by alternative means.

4. **Performance**

- The system shall be available from all network connection types used within the enterprise.

5. **Supportability**

- The developed system shall not use proprietary XML tag sets or languages.
- To the maximum extent possible, content viewing shall not use proprietary APIs. Server-side development and processing tools shall be best of breed within the XML community for accuracy, speed, and least life-cycle costs.
- Namespace and design patterns shall adhere to Federal and DOD policy guidelines when possible (to ensure the maximum interoperability possible).

6. **Implementation Constraints**

- Selection of server hardware and software shall be consistent with the knowledge base existing at the implemented hosting location.

7. **Interfaces**

- Media files shall be de-referenced with hyperlinks to prevent slow download times of the media's metadata.

8. **Domain (Business) Rules**

- Users can add, modify, and delete data only to the online version of the system. Data downloaded for a connectionless setting will only have viewing capability.
- A person may or may not be associated to a resource or activity. Creation of the association will only be available upon creation of the person within the system. A person may not be associated with more than one resource or activity.
- Users shall not be able to modify comments from other users.
- Comments shall only be created about existing content - comments may not be created referring to other comments or votes. This rule

applies to the referential integrity rules of the data, and not the author of the text.

- When performing a vote of quality on existing content, no data will be stored as to who voted (person, location, number of votes, etc.)

9. Legal Issues

- Sufficient warnings shall be present to communicate the level of security available to the system for both information entry as well as sharing to public and federal entities outside of those who have access to the system.
- Sufficient warnings shall be present to ensure that inappropriate remarks are not entered into the system, especially those that violate laws, regulations, or policies.

F. REQUIREMENTS GLOSSARY

> [T]he glossary is a list of noteworthy terms and their definitions...The goal is not to record all possible terms, but those that are unclear, ambiguous, or which require some kind of noteworthy elaboration, such as format information or validation rules.[45]

<ContentNodes> - an element within Location.xml files, which contains a collection of **<ContentNode>** elements that refer to content types that are available (or that have been previously created) for a corresponding Location.

AvailableContent file - a collection of content nodes within an XML file with descriptions on their use in the system. For example, a Routes.xml content file is a collection of Route content nodes, and a People.xml content file is a collection of Person content nodes. Content files are 'local' to a specific instance of a Location content node (i.e. Routes or People belong to only one Location).

Content node - an object of knowledge within the system, which can be validated against an XML Schema. It is a logical collection of data and information elements. For example, a particular instance of a route, pier, or person is a content node, and contains related data and information elements (see also <ContentNodes>).

Dummy node - an XML file that contains one empty element. Dummy nodes are used in the creation of blank HTML forms.

folderName - the name of the file directory (or folder) of a location. The exact name is the <LocationName> element's contents with all formatting and white spaces removed (e.g. the folderName of "St. Charlotte's Bay" would be "StCharlottesBay"). A numeric value may be added to the folderName in the event that another folderName already exists.

Hidden data - data transmitted within a HTML form, contained within HTML <input type="hidden"> elements that are intended to be used by the system and not the user.

id [also nodeId] - 1. In HTML, it is a unique identifier of an element tag within a HTML document. It must not start with a letter and may contain only the valid characters as specified by the W3C's HTML 4.01 Recommendation[20]. 2. In the 'XKMS' system,

it refers to the time-stamp identity of a particular content node (the number of milliseconds since the a computer operating system's time epoch). These time-stamp id's are used throughout the system in creating HTML id's by pre-pending them with character sequences. The id's also correlate to a similar meaning as a 'key' field in a database.

nodeAction - refers to an action to be taken (Add, Modify, Delete, etc.)

nodePath - refers the relative path URL to a particular Location within the system.

nodeType - refers to that specific class of content node (e.g. Location, Route, Resource, etc.).

refId (Reference id) - is the 'id' of a related content node. It is used to relate content nodes to one another (e.g. People to Resources or Activities, and Comments to any node type). The use of the refId is similar to that of a 'foreign key' in typical database schemas.

xkms – XML-based Knowledge Management System.

xkmsAuthor - is a server Session variable, which is a text string of XML representing an Overhead node. This variable is a container that holds the information input from the logon form.

xkmsReceiver - is an IIS Application Server Page (ASP) that receives SOAP messages, processes the messages, and replies with a SOAP response message.

xkmsTransmitter - is an IIS Application Server Page (ASP) that receives input via URL search string and fires a HTML form to the client using XSLT and DOM.

THIS PAGE INTENTIONALLY LEFT BLANK

III. INTERNET AND XML TECHNOLOGIES

A. INTRODUCTION

The purpose of this chapter is to provide an introduction to the technologies used, their significance, problems discovered, and compare them to other database technologies. Figure 3 graphically illustrates the relationships between each technology. The technologies used in developing the system are grouped into three categories: World Wide Web Consortium (W3C) technologies, the JavaScript language, and Microsoft Client/Server software (with W3C Application Programming Interfaces (APIs) and proprietary extension APIs). While specifically desiring not to develop the application using proprietary APIs, use of Microsoft technologies were necessary because it matched the author's existing skill set and the Coast Guard's enterprise-wide software available.

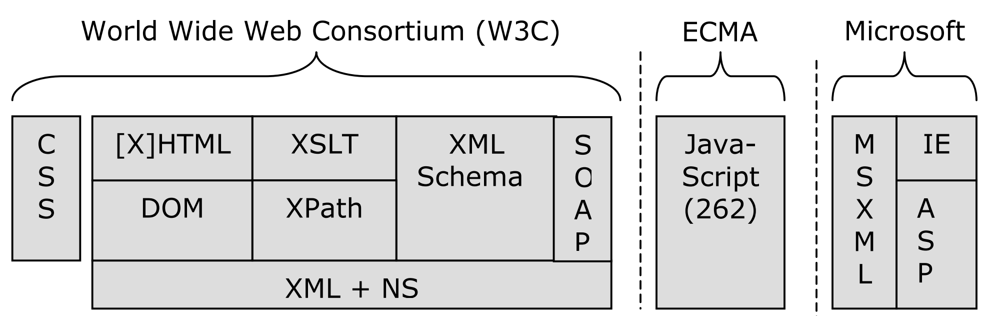


Figure 3. Technology Relationships

B. XML (EXTENSIBLE MARKUP LANGUAGE)

XML became a W3C Recommendation in February 1998[15]. It is a subset of the Standard Generalized Markup Language (SGML)[14]. XML is about structuring data in a meaningful way and using user defined tag sets that describe the data within them [21]. It can be used to create data stores and other languages; however, it is not a language itself. XML is plain text (usually in UNICODE format), which is common to most every computing system in the world. With text as a common baseline, and standards that are non-proprietary, XML related technologies are becoming prevalent in many aspects of computing.

XML was used to store all of the information within the system. An individual XML file in the system acts in a similar way a table would in a relational database system. While the central focus was XML, little of the capabilities contained in the XML Recommendation were used. XML Schemas were used in lieu of Document Type Declarations (DTD) due to their ease of construction within the XML Spy IDE. XML Schemas also have greater robustness in validating complex data structures. Little use was made of predefined XML entities. The only difficulty encountered in using XML was in the differences between XML and HTML predefined entities (e.g. * * is valid HTML, but not defined in XML).

Though not part of the XML Recommendation, the importance of XML Namespaces cannot be overlooked when discussing XML[28]. XML Namespaces are used to associate element and attribute names to a specific URI (e.g. creator or owner); however, they were not used in the application. This was intentional - it kept the design more manageable during the numerous changes to content nodes. It also simplified the processing of DOM XML documents. Any future implementation of this system should use XML Namespaces.

C. XML SCHEMA

XML Schema became a W3C Recommendation in May 2001[43]. Schemas are sometimes referred to as 'instance' documents about other XML documents. Schemas can establish the structure of elements and attributes within XML documents, including their cardinality. Schemas also define data types that both elements and attributes may contain. XML documents may contain references to XML Schemas within them, or they can be programmatically applied within an application (as was the case for the developed application). XML Schemas may even be contained within the XML document containing the data it enforces.

XML Schemas were used to define and enforce data typing of all content within the system. As mentioned previously, the alternative of using DTDs was not chosen. XML Schema performs many of the same functions as SQL type schemas, but differ by the unique structure of XML documents.

XML Schema validation is part of the MSXML parser. However, XML Schema validation is only available in version 4.0[26]. This proved to be problematic due to the school's computers not being loaded with this version. However, client side validation is possible with small modifications to the JavaScript executed by the client (i.e., the functionality is currently delimited as comments).

The complexity of XML Schemas is so vast that the W3C Recommendation is broken into three parts, the first being a primer[43]. One particular difficulty in using XML Schema is understanding the difference between simple and complex types (simpleType and complexType). The differences between them affect the manner in which data types can be validated. This led to design changes in the content schema for the developed system that are not necessarily obvious without this knowledge.

D. XPATH (XML PATH LANGUAGE)

XPath became a W3C Recommendation in November 1999[42]. "XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer[42]." In other words, it is nearly impossible to perform an XSLT without using XPath. XPath was used to access data from content files within the developed system.

XPath is not equivalent to SQL. While XPath is very powerful at reaching into an XML document and finding nodes or node sets, it is not a complete query language. For operations like joins, unions, intersections, etc., SQL is far more capable. Thus, the developed system is inherently weaker than existing relational database systems.

The XPath standard is likely to have an ever-increasing presence. For example, future DOM standards are being completed to implement XPath functionality in navigating DOM documents[11]. XPath concepts can potentially be extended to other areas where objects exist in a hierarchy of similar type objects.

E. XSLT(EXTENSIBLE STYLESHEET LANGUAGE TRANSFORMATION)

In the historical order of W3C Recommendations, the first standard for transformations was the W3C Recommendation titled "Associating Style Sheets with XML documents" of June 1999[1]. This recommendation created the <?xml-stylesheet?> processing instruction which triggers the XML parser to apply a stylesheet operation (or

transformation) to an XML document. Next came XPath and XSLT in November 1999[42, 47]. XPath was described in the previous paragraph. XSLT provides a set of declarative programming language instructions to transform XML documents into other types of documents (including XML and HTML). More recently, the Extensible Stylesheet Language (XSL) became a Recommendation in October 2001[16]. The XSL provides the ability to format the layout and presentation of documents as outputs of a transformation process (similar to printed document typesetting). Unfortunately, these Recommendations can be confusing. First, XSL files are built on top of the XSLT technology. Therefore, XSLT documents (*.xslt* files) are valid XSL documents (*.xsl* files). Second, XSLT documents are commonly referred to as "XSL Transformations" and XSL documents are commonly referred to as "XSL Formatting Objects (XSL-FO)". For each document type, there requires at least an XSLT processor and possibly an addition XSL-FO formatter.

The XSLT processor is the most used tool within the developed system. XSLT processing is part of the MSXML API. It is used to transform data files into user views, and to create forms for data entry, modification, and deletion. XSLT can load multiple documents to create dynamic associations between separate but related data. XSLT can sort, format, and perform calculations on data during the transformation. Finally, XSLT can provide outputs usable in other XML operations or as HTML files to be served to a client. In comparing XSLT to other database systems, the XSLT and its processor perform some of the functions of the runtime subsystem and the data engine.

While XSLT can do many tasks, it comes by way of a steep and long learning curve. When coupled with XPath, navigating a document to produce the output you desire can be a challenging task. The difficulty in learning the XSLT language, especially regarding the use of variables in recursive tasks, is greater than with relational database systems. The author learned XSLT mainly in a trial-and-error process.

F. DOM (DOCUMENT OBJECT MODEL)

DOM became a W3C Recommendation in October 1998[9]. "W3C's Document Object Model (DOM) is a standard Application Programming Interface (API) to the structure of documents; it aims to make it easy for programmers to access components

and to delete, add, or edit their content, attributes and style[12]." The current and future DOM modules are the Core, XML, HTML, Events, Cascading Style Sheets, Load and Save, Validation, and XPath. These modules are being developed in a phased approach, as levels. Each of the DOM Levels may have numerous documents regarding specific modules. Figure 4 graphically communicates the coordination between the numerous DOM components. DOM Level 3 is the last of the schedule developments.

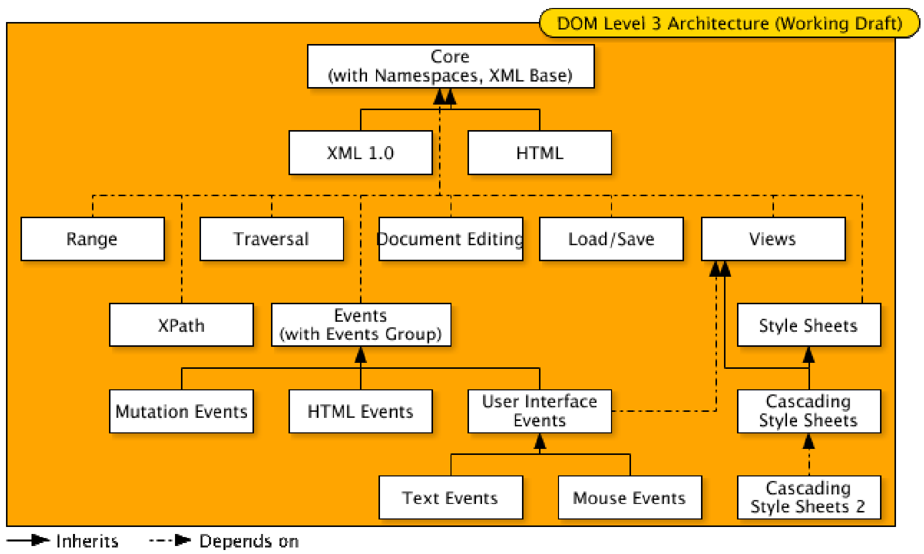


Figure 4.   W3C's DOM Architecture
(From: http://www.w3.org/DOM/Activity, 25 Jan 03, W3C (c) 2003)

In the developed system, DOM XML and DOM HTML were used to programmatically manipulate XML and HTML documents. DOM provides the functionality that XSLTs cannot, or that are more efficiency performed with DOM. The DOM application programming interfaces (APIs) are part of the MSXML Component Object Model (COM) Dynamic Link Library (DLL). Both were accessed through Microsoft's JScript (JavaScript) language at both the client and server. At the client, DOM HTML was used in creating dynamic behaviors of views. At the server, DOM XML was used to manipulate specific content tags.

Two important facets of working with DOM are its similarities to other XML technologies and the manner in which DOM Levels are formalized. First, DOM and XSLT both represent XML documents in a hierarchical manner, with only minor differences in their node representations. However, working with XML documents vary greatly between DOM, an imperative programming API, and XSLT, a declarative language. The differences between the two models affect the manner of program execution. The order in which XSLT statements are executed is implied by the data source (similar to how SQL statements are executed), and not on the sequence of programming code.

Second, many of the features in the MSXML parser have yet to be formalized by the W3C. For example, the 'standard' to load and save XML documents has yet to be set[10]. This lends itself to use of tools that may be deprecated as new standards emerge. The notion of using a vendor's API before a standard is agreed upon is not new. There appears to be a cyclical nature of developers to extend the technology with proprietary features, which are agreed upon in the future as W3C Recommendations.

Additional information on the history of DOM is discussed in Chapter IV. Chapters IV and V cover many more details on the use of DOM, including problems and solutions that were found.

G. HTML & XHTML (EXTENSIBLE HYPERTEXT MARKUP LANGUAGE)

The latest version of HTML (version 4.01) became a W3C Recommendation in December 1999[20] and XHTML became a W3C Recommendation in January 2000[36]. XHTML is a reformulation of HTML in XML. That is, XHTML is valid XML. However, the W3C Recommendation for XHTML covers many other minor issues to improve longstanding problems with HTML.

HTML was used as the presentation medium for all interactions with the system. The HTML documents were the result of applying XSLTs. While the results of the transformations were valid XML, they were only partially valid XHTML documents. Though the *name* attribute is valid HTML, it has been deprecated in XHTML in lieu of using the *id* attribute. The author chose to continue using the *name* attribute due to inconsistent behavior in how Microsoft JScript treated the *id* attribute in form elements.

H. SOAP

SOAP was used as a communication message format between the client and server. SOAP version 1.1 only reached the status of a Note within the W3C. SOAP version 1.2 is currently a Candidate Recommendation (until Jan '03) and will no doubt become a full Recommendation[31].

The notion of SOAP is simply the use of an electronic envelope which may contain header information to describe the envelope's content and the body of the envelope with contains the XML sent from one point to another. The complexity of SOAP is in its definition of versioning, error reporting, intermediate node processing protocols, and extensibility. The author attempted to fully comply with the SOAP Recommendation in developing the system, one exception was omitting XML Namespaces within the SOAP envelope.

SOAP is an important component to the implementation of web services. SOAP is a messaging format for use with the Web Services Definition Language (WSDL). WSDL is a separate standard being developed by the W3C[33]. Interoperability between systems is potentially possible with the use of SOAP messaging as described in a WSDL service. The process of sending and receiving SOAP messages is described in Chapter V.

I. CSS (CASCADING STYLE SHEETS)

CSS became a W3C Recommendation in December 1996[8]. Cascading style sheets is not an XML technology. However, the use of CSS is invaluable to the use of presenting XML content. CSS is a language to express the formatting and layout of a HTML document. Coupled with DOM HTML (or dynamic HTML), it provides powerful features to organize and present content to the user. The development of large-scale web applications is far simpler with CSS.

Every view the system produces uses a single *.css* file. CSS enabled:

- A consistent format to text and colors across all pages within the system (i.e., template).
- The ability to typeset blocks of text within the browser's window.
- The ability to dynamically show, insert, and hide data from the user.
- The ability to simulate hyperlinks for script function calls using in-line text elements.

More specific style and behavioral controls were added to individual HTML tags on a case-by-case basis.

J. MSXML (MICROSOFT XML PARSER)

Microsoft's XML parser was used as the only XML parsing tool within the system. MSXML's latest version was released in October 2001 (version 4.0, Service Pack 1)[26]. The latest version of the MSXML parser provides a robust feature set, including many features that are still being worked on by the W3C. Most notably are the means to persist XML documents (i.e. save them to disk) and the ability to validate XML files against an XML Schema.

One of the problems encountered by the author with the MSXML parser is the side-by-side installation of newer versions of the API[26]. Newer versions are installed without replacement of older versions, which can dramatically affect the expected outcome of the application under development. For instance, *MSXML2.DOMDocument* refers to the DOM XML document COM object, and is valid as-is. However, unless the developer appends the latest version (*.4.0*) to the object declaration, version 3.0 is used which does not support the latest processing features (including XML Schema operations). This "feature" was intended to protect legacy applications from being affected by changes to the API[26]. One could argue that this strategy is flawed and is not strategic in nature.

Another problem encountered is the variability of which MSXML parser is installed on computers. Internet Explorer version 5.5 may have either MSXML Parser version 3.0 or version 3.0 service pack 2[26]. Windows 2000 may have either MSXML version 2.0 or one of several service packs of version 3.0. Windows XP and Internet Explorer ships with version 3.0 service pack 2. Because of the pressure to release

updates to the MSXML parser out of sync with newer operating systems and browser software, versioning will be a critical requirement for any implementation using the MSXML API. As a case in point, unless the client computer is upgraded to MSXML version 4.0, XML Schema validation cannot be accomplished.

K. INTERNET EXPLORER (IE)

The author developed the application using Internet Explorer version 5.5 and 6.0 as the target user agent. Each of these versions has a different 'built-in' version of the MSXML parser, which caused difficulties as described above. "Currently, the default XML parser for Internet Explorer is MSXML 2.0 or MSXML 3.0, depending on the version of Internet Explorer. Until Internet Explorer ships with MSXML 4.0 or later as its default XML parser, the MSXML 4.0 features are available in Internet Explorer only via scripting when an XML DOM object is instantiated using the appropriate version-specific ProgID. (http://msdn.microsoft.com)" The previous statement means that access to the latest version of the MSXML parser is via ActiveX objects. The use of ActiveX objects has two significant implications. First, ActiveX is proprietary and not supported by other browser vendors, thus using MSXML locks the developer into using specific vendor software. Secondly, ActiveX objects have received notoriety in the past as being high-risk security weak points. Thus, many operating systems' administrators have blocked the use of ActiveX objects from Internet Explorer. This aspect seriously impinges upon the potential implementation of this technology in the enterprise.

L. ACTIVE SERVER PAGES (ASP)

Active Server Pages are the server-side files that interface with the Internet Information Server (IIS) within the Windows operating system. Both ASP and IIS are Microsoft products. ASP files can be authored using scripting languages to control the outputs of the server. ASP files were developed to act as the database application engine (similar to that of the application system in other standalone database applications). Accessing the MSXML parser at the server was performed via ASP pages (with JScript).

The author intentionally designed the system so that it potentially could be re-programmed in an Apache Tomcat environment using Java Server Pages (JSP). As future XML technologies emerge, the leader in parsing and processing tools may come from an open-source Java-based provider. Future research regarding the portability of this application to such an environment is recommended.

M. JAVASCRIPT (ECMA STANDARD-262)

The European Computing and Manufacturing Association's ECMAScript language's 3rd edition was standardize in December 1999 (as ECMA Standard-262)[13]. ECMAScript is JavaScript[13] (or JScript as it is referred to in Microsoft development IDEs[23]).

JavaScript was used as both the client and server side scripting language. The scripting language performed the functions as discussed in the previous paragraph and as a function similar to that of Visual Basic for Application (VBA) does for the Access database application environment. JavaScript's implementation of the *try-catch* statements provided a significant advantage over VBScript in trapping and troubleshooting development errors.

One significant problem discovered in using JavaScript at the server is its inability to use the *variant* data type. The *variant* data type (a weak data type) is an integral part of the IIS environment. When binary files are uploaded to the server, they are treated as the *variant* data type. However, there is no such data type within the JavaScript language. Therefore, the implementation of the feature to upload files required the use VBScript in two of the ASP files.

IV. PROOF OF CONCEPT CONSTRUCTION

A. OVERVIEW OF DESIGN (LOGICAL PERSPECTIVE)

1. Purpose of the Proof of Concept

The purpose of building a proof of concept system was to demonstrate, in small scale, that a larger system was possible, to build the skills necessary to accomplish future work, and to identify problems to resolve before continuing. In essence, the proof of concept project was an initial iteration of both learning and problem clarification.

At the heart of the proof of concept was building basic functionality of the database system, the ability to view, add, modify, and delete data. Secondary goals were to ensure the separation of storage and presentation of data, and to ensure that the data was deployable to a connectionless setting. This system was built from a very specific data set.

The proof of concept required four weeks to construct. Most of the development time was spent learning, troubleshooting, and considering the various possibilities of design. This chapter explains the design and schema, development methodology, tools used, specific techniques, and problems identified. Some of the topics discussed in this chapter were not included in the prototype system due to improvements made to the design of the system.

2. Types of Data and Its Integrity

Numerous processes manipulate the data and information entered in the proof of concept system. The most common data type is text characters (including character data (CDATA)). Everything entered into a HTML form is text. During form validation, it may be converted into a numeric value or other data types, but once it is transmitted to the server, it becomes text characters again. Likewise, if the data is validated at the server, the character sequences will be parsed again into other data types (integers, floating point, strings, dates, etc.). Finally, in the process of serializing the data into XML, it is again treated as text characters. Any further validation performed by an XML Schema is performed within an XML parser.

Data validation in the proof of concept system was completed at the browser before transmitting it to the server using client-side scripting. As a tertiary design criterion in the proof of concept system, browser level data validation had to be consistent with the type of validation that could be performed with an XML Schema. This requirement leads to empty string, numeric range, and regular expression enforcement. Regular expressions are a significant and powerful feature of JScript. They provide a very robust tool to enforce data types within a character-based sequence and are similar to the capabilities of the XML Schema in the MSXML parser used. Unfortunately, the RegExp object is a Microsoft proprietary extension to JavaScript[26]. Thus, the proof of concept application only functioned correctly in Internet Explorer.

3. Stored Data and Information

The primary record within the proof of concept application is a geographic location, which is stored within an XML file named *Location.xml*. The data and information contained within each *Location.xml* record is listed in Table 1. With the exception of the *Description* field, all other fields are small snippets of data. The *Description* field is unique in that it is an unbound character data source where the user is able to enter their own perspective and thoughts of how to describe the location (hence, an information field). The use of these types of entries is prevalent throughout the system.

The *ContentNodes* element within a *Location.xml* file is a container that holds references to information nodes related to that location. A specific *ContentNode* points to a specific content file of a specific type (e.g. routes, piers, anchorages, etc.), this is similar to the manner in which a key field would 'point' to a related record in an E-R type database.

Table 1. Location Data Schema

Field	Node type	Contents
Location	Element	CDATA
tagName	attribute	Text, [child of Location]
type	attribute	Text, [child of Location]
timeZone	attribute	Text, [child of Location]
obsDaylightSavings	attribute	Text, [child of Location]
Description	Element	CDATA
Coordinates	Element	None
datum	attribute	Text, [child of Coordinates]
Latitude	Element	Text, [child of Coordinates]
Longitude	Element	Text, [child of Coordinates]
UNCTADLocode	Element	Text
CoastPilot	attribute	Text, [child of CoastPilot]
SailingGuide	attribute	Text, [child of SailingGuide]
ContentNodes	Element	None
ContentNode	Element	CDATA, [child of ContentNode]
type	attribute	Text, [child of ContentNode]
href	attribute	Text, [child of ContentNode]

Tables 2 and 3 describe the relational and administrative data needed to maintain the proof of concept system. The design of the relationships is described later in this chapter. The administrative data keeps information on who authored the content, when it was authored, and the evaluation of users providing feedback on the quality of the content. The data contained in Table 2 was dropped in the prototype design due to its redundancy. An improved XSLT was created to eliminate the need for that duplicity.

Table 2. Location Relationship Data

Field	Node type	Contents
LocationLinks	Element	None
LocationLink	Element	CDATA, [child to LocationLinkS]
type	attribute	Text, [child of LocationLink]
tagName	attribute	Text, [child of LocationLink]

Table 3. Location Administrative Data

Field	Node type	Contents
Overhead	Element	None
timestamp	attribute	xs:DateTime, [child of Overhead]
Author	Element	None, [child of Overhead]
Name	Element	Text, [child of Author]
email	attribute	Text, [attribute of Name]
PhoneNumber	Element	Text, [child of Author]
type	attribute	Text, [attribute of PhoneNumber]
QualityRating	Element	None, [child of Overhead]
avgValue	attribute	Number, [child of QualityRating]
votes	attribute	Number, [child of QualityRating]
Association	Element	CDATA, [child of Overhead]
type	attribute	Text, [child of Association]
href	attribute	Text, [child of Association]

4. Relationships Between Data

The collection of locations is stored in a hierarchy of system folders (or directories) on the web server. Sub-locations correspond directly with their actual geographical relationship to a parent location. For example, the State of California is part of the west coast of the United States, which is a part of North America.

The referential relationships between locations are summarized as follows:

- A parent location has information about its children
- A child location has information about its parent (but not its siblings)
- The 'index' globally has information about every location.

The above criteria were chosen to keep the design of the system simple. The ability to have knowledge about siblings, decedents, and ancestors was solved and further described in the prototype, Chapter V.

B. DATABASE ARCHITECTURE (PHYSICAL PERSPECTIVE)

1. Non-XML Technologies Used

In development of the proof of concept system, the following non-XML technologies were used:

- Application Server Pages (ASP) – the script environment contained within Microsoft's Internet Information Services (IIS) web server. Through ASP you can access the following Internet Server Application Programming Interface (ISAPI) objects: Application, Request, Response, Server, Session, ObjectContext, and ASPError.
- ECMAScript (JScript/JavaScript) – originally created by Netscape, this scripting language is used within ASP and client web pages.
- Document Object Model (DOM) –DOM originated itself as a standardized version of the commercial 'dynamic' HTML APIs (DHTML or dHTML)[23]. DOM Level 1 is generically geared for both XML and HTML. As an appendix, the DOM Level 1 recommendation provides a sample ECMAScript API. And while that API is not part of the ECMAScript standard, it has been implemented as an object within Microsoft's JScript environment.
- Cascading Style Sheets (CSS) – "is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents[8]." CSS provides standardized and centralized formatting for the viewable portion of the system. A CSS is simply a reference (embedded or as an external file) to a statement of formatting. The use of CSS within the system is discussed in more detail later in this chapter.
- FileSystemObject (FSO) – "allows you to use the familiar *object.method* syntax with a rich set of properties, methods, and events to process folders and files[26]." While the MSXML API provides for the creation and saving of XML files, it neither provides a means to delete nor a means to create or change folders. The FileSystemObject is provided within the ASP environment for such activities. The limitations of the activities that can be performed are tightly coupled with the Windows NT File System (NTFS) permissions for the appropriate folder. This topic is discussed in more detail later in this chapter.

2. XML-Related Technologies Used

In development of the proof of concept system, the following XML technologies were used:

- Extensible Markup Language (XML) – described in Chapter III.
- Microsoft's DOM XML – as both an implementation and extension to the current recommendations from the W3C, the MSXML DOM allows XML

Schema, XSLT, and XPath operations, in addition to simple operations not formalized by the W3C like loading and saving XML documents (i.e. persistence). Access to the MSXML API is through an ActiveX object. ActiveX objects are from the same family as COM and Object Linking and Embedding (OLE) objects, and thus are proprietary implementations.

- Extensible Stylesheet Transformations (XSLT) – described in Chapter III. XSLT is used extensively in viewing XML documents. It is also used for 'preparing' data produced by one means for consumption by another.

3. Proof of Concept File Use and Layout

Table 4 lists the types of files developed for the proof of concept and their intended use. Note that there are actually two different types of *Location.xml* files. The first type is contained the */World* folder. This file contains data on every location within the system. This 'index' *Location.xml* file has its own viewing transformation named *LocationMapper.xslt*. All other *Location.xml* files use the *LocationView.xsl* XSLT for creating a view of the information. The design of separate add, modify, and delete files for each type of content is a common characteristic in web application design. Unfortunately, that methodology required significant amounts of programming code when using the DOM XML API. Out of necessity for efficiency, a new method was used in the prototype application which is explained in Chapter V.

Table 4. List of Files in Proof of Concept System

Location.xml – contains the system's index of locations
LocationMapper.xslt – used for viewing the index

Location.xml – the standard location data file
LocationView.xsl – used for viewing a location's information

LocationAdd.asp – used for adding new locations to the system
LocationModify.asp – used for modifying existing locations
LocationModify.xsl – used to insert data into a HTML form
LocationDelete.asp – used to delete existing locations

Logon.asp – used to log a user into the system
Error.asp – used to report a processing error to the user

generic.css – used as a style reference for the above non-XML files

Figure 5 is the screen capture of the file system used in the proof of concept system. Notice that all Locations are descendants of the World at some level. The *World* is the base folder and contains the deployable portion of the system (that is, the content that could be displayed without an Internet server). One folder above the *World* directory is the *xkms* folder which contains the system files used to execute the non-deployable portion of the application (create, modify, delete, etc.).

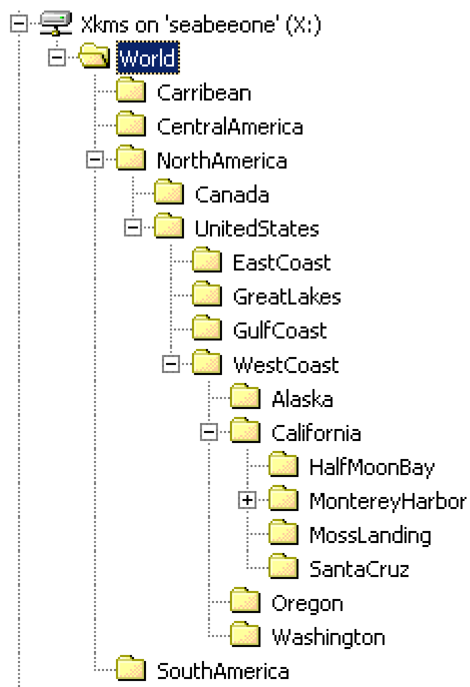


Figure 5. Physical File Structure

The relationship between files in the system is described in Figure 6. The connection between the *World* folder and all of its children is relative. That is, viewing the content within the system is independent of the actual location of the file, as long as a correct relative path Uniform Resource Locator (URL) is provided to the *LocationView.xsl* file.

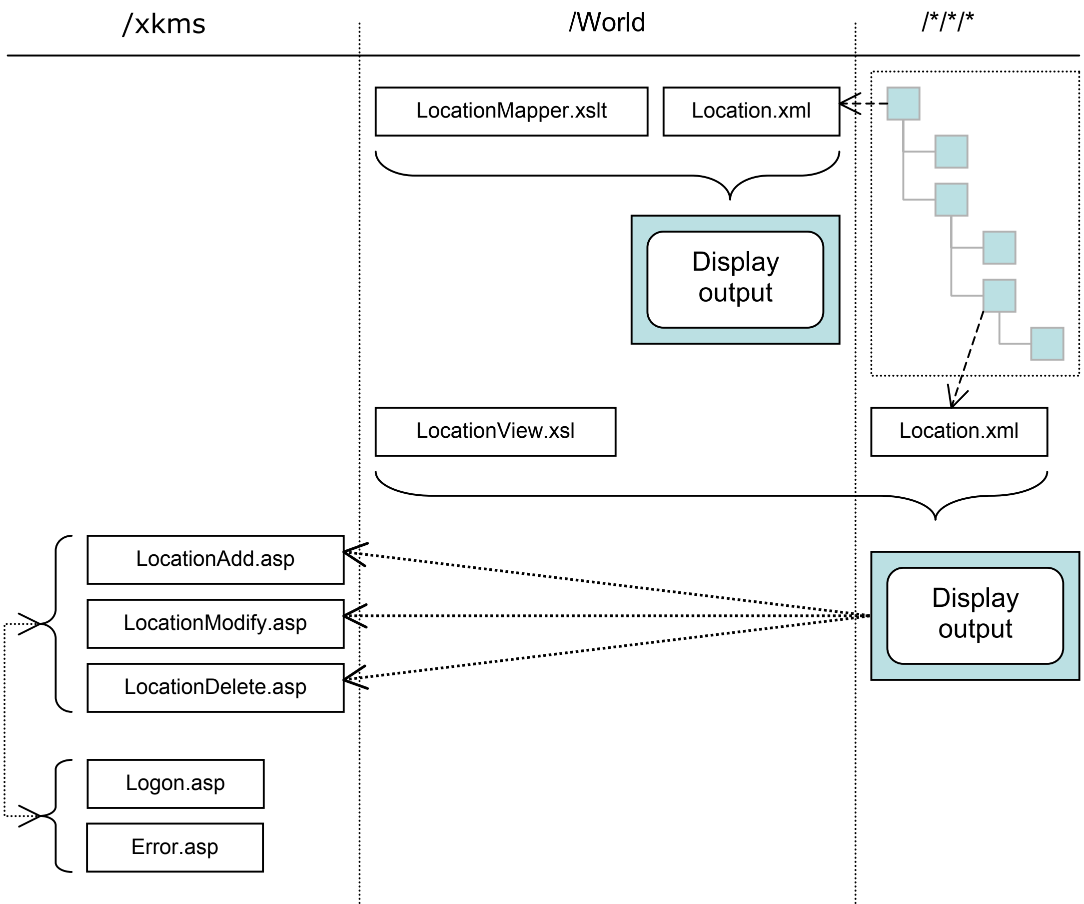


Figure 6. Proof of Concept File Relationships

C. APPLICATION CONSTRUCTION TECHNIQUES

1. Relative Path Navigation

Navigating a web site via hyperlinks can be performed by two path-naming conventions: absolute and relative. Absolute paths contain a full URL to a file. For example, an absolute path to a file would read as *file:///c:/myDrive/myFolder/myFile.htm*, and an absolute path to a web page would look like *http://www.mySite.com /myFolder/myFile.htm*. Both of these patterns follow the IETF protocols of *protocol // hostname [:port] [/ path]** [23]. A hyperlink may omit the portion of the URL where the browser (or agent software) is currently focused. From the previous example, if the browser is currently at *file:///c:/ myDrive /index.htm* and wants to view the contents of

myFile.htm, the relative URL path is *myFolder/myFile.htm*. Figure 7 illustrates the files used in this and the following examples.

```
File (file:///)                 Internet (http://)
C:/myDrive                      www.mySite.com
  ↳ index.htm                     ↳ index.htm
  ↳ myFolder (folder)             ↳ myFolder (folder)
      ↳ myFile.htm                    ↳ myFile.htm
```

Figure 7. URL Path Navigation Example

One of the benefits of relative URL navigation (or hyperlinking) is that the same navigation can be performed using various types of protocols (e.g. http, ftp, file). From the examples above, the relative URL path *myFolder/myFile.htm* can be used for browsing files on a disk drive or from an Internet server. Even though Microsoft-based operating systems represent folders with backslashes (\), Internet Explorer automatically converts a HTTP path's forward slashes (/) to the proper format. This functionality is critical to the system's design. It allows the downloaded data to be viewed and navigated in a connectionless setting in the same manner as the Internet server environment.

An important related topic to relative URL navigation is the ability to navigate upward in a file structure as well as downward. Downward navigation was illustrated previously. Upward navigation is possible through the use of a special character sequence '..' (two periods). The '..' references the parent of the current directory (or folder). Continuing with the previous examples, if the current focus is *myFile.htm* a relative path hyperlink back to *index.htm* would look like *../../index.htm*. The reference would read as, "go to parent folder of myFile.htm (myFolder), go to parent folder of myFolder (www.mySite.com or myDrive), and get index.htm." In addition to the '..' character sequence, there is also the '.' character sequence, which references the current directory. As a relative path, *myFolder/myFile.htm* and *./myFolder/myFile.htm* perform exactly the same. Except in certain Unix-based environments, the current directory reference is unnecessary.

The system uses relative URL paths for navigation. All content is based from the *xkms* folder. All viewing related XSLT files reside in the *xkms/World* folder. The viewing XSLT files and their respective content files have a one-to-many relationship (1:N). Therefore, similar content files, regardless of their depth within the system, all need to reference the same viewing XSLT. The relative path URL is computed at creation of the content file, and then inserted as a processing instruction into the XML file, similar to the following: <?xml-stylesheet type="text/xsl" href="../../../../Location View.xsl"?>.

2. Dynamic Information Exchange

A critical component of any web-based application is its ability to exchange data dynamically. The alternative is a static set of view-only web pages. The history of dynamic web content began with the Common Gateway Interface (CGI). CGI was developed at the National Center for Supercomputing in 1983 for the HTTP Daemon server[18]. CGI acts as a common layer between web servers and other applications (including compiled and script languages)[7]. Microsoft's version of CGI is the Internet Server Application Programming Interface (ISAPI)[26]. While both of these interfaces provide robust features, their typical use is to translate HTML form and URL data into useful data for applications.

As an example, a full path URL might look like the following: *http://www.myServer.com:80/folder/file.asp?name1=value1&name2=value2#location*. The protocol is *http*, the host name is *www.myServer.com*, the HTTP port number is 80, the path information is *folder/file.asp*, the query string is *name1=value1&name2=value2*, and the hash value is *location*[23]. Everything appearing after the port number is handled by the interface (e.g., CGI, ISAPI, etc.).

Web-based information passing is most commonly performed using the Hypertext Transfer Protocol (HTTP) *Get* or *Post* commands to a target file such as an Active Server Page (ASP) or Java Server Page (JSP). With a *Get* command the information is passed in the URL as a query string. The query string is parsed into *name-value* pairs for use by dynamic content applications. With the *Post* command, the information is embedded into the HTTP body, but still in a *name-value* pair format. The question mark character (?) is the signal that a query string follows[6]. Depending on which interface is used, white

spaces and illegal characters are escaped in the query string. For example, the addition character (+) or a hexadecimal ASCII reference (%20) is used to represent a space character, depending on which interface is being used.

Hash values (or bookmarks), or what appears following the ampersand (#) character, in the URL string are generally used to navigate to a specific location within a web page. Browsers will navigate to the HTML element tag that matches the tag's *id* attribute (e.g. <a id="lookHere"/>). This is a useful feature for locating a specific location within a long document.

In the designed system, CGI-style data passing is a critical feature. For example, if a user desires to add a sub-location to the location they're currently viewing, there needs to be some mechanism to communicate where to add the new *Location.xml* file (similar to variable passing). That communication is performed using the URL query string. From the existing web page, a JavaScript procedure reads the current browser's location (using the JavaScript Location( )) object and searches the current URL string for everything that follows the *World/* sub-string. The backslashes are then translated to an alternative character and appended to the hyperlinks using dynamic HTML (or DOM HTML) methods. For example, if the browser is currently focused on the file *intranet.server.mil/xkms/World/NorthAmerica/WestCoast/ California/SanFrancisco/Location.xml* and the user wants to add a sub-location, then the resulting HTML hyperlink would present itself as *../../../../../../LocationAdd.asp? Location=World+NorthAmerica+WestCoast+California+SanFrancisco+*. The above example URL uses a combination of relative path addressing (taken from the *xml-stylesheet* processing instruction) and a query string. The subtleties of the above are significant. First, the initial web page can be located anywhere on the server, and performance of the system is unaffected. Secondly, the initial web page doesn't store information about where it is in the system. The query string provides the target location to store the new information.

3. Cascading Style Sheets (CSS)

CSS provides the developer an ability to 'cascade' sets of rules to achieve a particular style to the presentation of a document. A CSS rule instance is described by a selector, which is followed by a set of style properties. For example, *body { font-family: Verdana; font-size: 10pt }* is a CSS rule[25]. The selector in this case is the HTML tag <body>. Three other common selectors are the HTML attribute *class*, the HTML attribute *id*, and the hyperlink behaviors (*link*, *visited*, and *hover*). The formatting capability of a particular style is very robust, similar to the typesetting capabilities of many word processors (e.g. Microsoft Word).

CSS rules may be embedded into a HTML document or referenced to a separate text file with the HTML <link> element. Within a HTML document, the influence of a particular CSS rule depends on the tag's selector and its hierarchical relationship of other HTML tags. For instance, a paragraph tag <p> within the HTML <body> tag will inherit the body tag's style, unless specifically set by another rule. Thus, immense complexity can be achieved with references to styles, the inheritance levels of styles, and the dynamic effects that can be controlled with the DOM interface.

4. ASP Files That Serve and Process Forms

How does the same ASP file create a form, and then have the *Post* function return to the same file for processing? This is achieved by using the HTML form element <input type="hidden" name="return" value="anything">. Upon each call to the same .asp file, a script *if-else* statement tests whether the Post-ed *name=value* pair *return="anything"* exists. If there is no Post-ed data, then a null value will be returned. This will be the trigger to create a form for the client to complete. Upon returning to the same page with the form's data in the HTTP request's body, there will exist a *name-value* pair corresponding to *return="anything"*, which indicates there is a completed form to process. At the end of the processing section of the .asp file, the processing script will need to redirect the client to another page using the Response.Redirect( ) method of the ISAPI. The alternative to this design is to create a static HTML file for form data collection, which *Post*s data to an *.asp* file as the receiver of the data. The need for this

type of HTML programming was eliminated in the prototype system through the use of SOAP messaging to the server.

5. Use of the ASP Session Object

The ASP Session object enables the client to maintain presence on the server while visiting different pages within the same site. In the proof of concept system two Session variables are created. They are named *xkmsFolder* and *xkmsAuthor*. These variables are actually instantiated objects, which are linked to the HTTP port number generated by the client browser (or agent). Once the client fills in the logon form, their data is stored as an XML string in the *xkmsAuthor* session variable. If the session variable is either empty or does not exist, then the client will be redirected to the *Logon.asp* file before any data manipulations are performed. The *xkmsFolder* Server session variable was not used in the prototype application.

6. XML Schema Datatypes

In general, data typing is specific to a particular development environment. The XML Schema recommendation contains a few unique formats for primitive data types, which is not directly compatible with many existing environments. As a baseline requirement, data stored within the system shall be in a XML Schema data type. Representing dates and times is particularly challenging. XML Schema *dateTime* type represents a date and time combination in a format compliant with the United Nations' (UN) International Standards Organization (ISO), Standard 8601[22]. In order to time stamp record creation within the system, a small algorithm was created to convert the data contained within an ECMAScript Date( ) object into the format required by the XML Schema Recommendation. While this provides uniformity, including during XSLT operations, any further usage in another application would require the text representation of the XML Schema *dateTime* to be serialized again. In essence, the XML Schema provides an independent party standardization of many data types in use across many development platforms.

7. Recursive Node Search Versus XPath

While proprietary technologies were required for successful completion of this system, a conscious effort was made to minimize their use, especially extensions to existing standards. As an example, the *selectSingleNode* method of the MSXML DOM implementation is an extension to the DOM Recommendation, and not supported in non-Microsoft environments. The *selectSingleNode* method returns a DOM node given an XPath expression. In order to perform the same operation with the existing DOM Recommendation, a recursive node search is required. While the search may increase development costs, it was done to separate the use of proprietary and non-proprietary DOM extensions.

D. DESIGN PROBLEMS AND TECHNOLOGY LIMITATIONS

In development of the proof of concept, many discoveries were made. The following sections discuss the lessons learned and limiting factors influencing the next phase of development.

1. Server Security

Powerful web server security is a difficult task. The task of defining and implementing varying user level access privileges was not developed for this system; it is left as a future research topic. As a consequence, the anonymous Internet user at the server has nearly full access to add, modify, and delete data files. This situation is not uncommon. A similar scenario is created when constructing a three-tier architecture using a database with an ODBC connection to the web server. Mitigating these risks remain somewhat of a low priority if the system is implemented within a controlled environment, such as a corporate Intranet. However, exposure to the Internet or intra-agency personnel staff will require this topic to be revisited.

2. W3C DOM, ECMAScript, and Microsoft

The history of DOM, its current status, and the future of its implementation into various platforms (JavaScript, Java, .Net, etc.) are complicated and confusing. The initial DOM Level 0 was meant as standardization between Internet Explorer's and Netscape Navigator's dynamic HTML competition. DOM Level 1 superceded Level 0 as the standard[12]. Yet more confusing, DOM Level 1 is actually made up of two

components, a core which applies to both XML and HTML, and a specific HTML section. DOM Level 2 has also been made into a Recommendation, and DOM Levels 3, 4, and 5 are still being developed[12].

Microsoft's Internet Explorer supports DOM Level 1. The DOM API is accessible via JScript or VBScript. However, this DOM is not the same as MSXML DOM. While both of these APIs support DOM, they are not compatible. For example, *document.documentElement* accesses the root node in a HTML document (HTML DOM), and *[objectName].documentElement* accesses the root node of an XML document (MSXML DOM), but neither object may be set to the other.

The lack of compatibility between Microsoft's DOM HTML and MSXML DOM is by design. The capabilities of the MSXML DOM are much more robust, and perform many of the operations that the W3C has yet to come to agreement upon. I argue that the differences and incompatibilities will be ironed out within the next few years as competition forces improvements all around.

a. Live DOM

A document represented as a DOM document is live. That is, any changes made to a node or node-set will have an immediate effect on the existing DOM document. This creates an additional factor to consider and learn from when using the DOM API. It is possible to unknowingly remove nodes from one document and place them into another document. For example, the following statement: *newXMLdoc.documentElement.appendChild(oldXMLdoc.documentElement.childNodes(2));* does not copy one node of a DOM document to the other, it removes the node and places it into the other. The desired statement would need to be as follows: *newXMLdoc.documenteElement.appendChild(oldXMLdoc.documentElement.childNodes(2).cloneNode(true));*. This 'feature' of the DOM API was the source of hours of frustration in developing the proof of concept system.

b. Areas Re-Worked

In addition to the topics identified for future thesis work, the proof of concept effort revealed numerous problems that required correction in the second iteration. The following is an abbreviated list of those topics:

- Shortening or resetting the server 'Timeout' behavior when data modifications are performed.
- Re-naming element attributes of 'type' and 'tagName' due to their non-descriptive nature
- Organizing a complete data checking and error reporting strategy
- Removing as many external references within a document in favor of more capable XSLT files that pull data from other XML files using the *document()* method.

E. SUMMARY

As stated in the beginning of this chapter, the purpose of building a proof of concept system was to demonstrate, in small scale, that a larger system was possible, to build the skills necessary to accomplish future work, and identify problems to resolve before continuing. That effort was largely successful. The topics discussed in this chapter are revisited in the prototype system as a second iteration towards improvement of design and construction.

V. PROTOTYPE APPLICATION

A. INTRODUCTION

1. Prototype As a Second Iteration

The work accomplished in developing the prototype application can be viewed as a second iteration. Like the proof of concept application developed and described in Chapter IV, many problems were discovered and solved in building the second application. The important problems are discussed in this chapter.

The construction of the prototype application required approximately 8 weeks. As mentioned in Chapter IV, one of the secondary learning objectives was to attempt to abstract the processes within the application so that the overall system may be extended to other communities of knowledge. In large part, the prototype application proves that possibility and is discussed later in this chapter.

2. Design Carryovers

The physical perspective (or file layout) of the proof of concept application constructed was essentially re-used in the prototype application "as-is". The hierarchical file structure with one main index *Location.xml* file was restructured for increased functionality without changing its role or basic purpose. JavaScript and Cascading Style Sheets from the proof of concept application were enhanced to provide improved interfaces and controls.

3. Design Changes

Many modifications were made to the schema of specific *Location.xml* files. The changes were necessary for improving element names and attributes, to reduce duplication of data, and to remove items having little value to the system. For example, the attribute *tagName* was changed to *folderName* to better match its actual use in the system, and all *LocationLink* elements that described parent and children locations were removed with the use of a more capable XSLT. The name of a specific location was moved into its own tag set (i.e. *LocationName*) due to the limitations of *simpleType* structures within XML Schemas - *complexType* structures in XML Schemas may not have minimum and maximum character lengths. Lastly, each location was given a

unique identification (id) value, which eliminated the need to perform recursive searches of the main location index and enables the ability to quickly derive the parent, children, and siblings of a location in the viewing XSLT.

The most significant change between the proof of concept and prototype applications is the manner in which HTML forms are created and their data is processed. The proof of concept required separate (yet duplicate in many ways) forms for adding and modifying data. In the prototype application, both actions are rendered using the same XSLT. Another weakness of the proof of concept is that each individual element in the content file was created with specific (and unique) DOM statements at the server. This is a huge programming inefficiency and was replaced with an iterative loop and XSLT at the client. That pattern is further discussed later in this chapter.

B. CREATING DATA VIEWS

1. Data View Development Process

Data views are created using XSLT. For each type of content there exists one related XSLT, which creates a HTML view page of the data. Creating a data view by writing an XSLT from scratch is very difficult unless you have expert knowledge about how HTML is rendered in a user agent. To lessen the difficulty, the following process was used. Though the process requires more physical labor, it simplifies the complexity of the task greatly:

Step 1: Create a physical sketch of the data to be presented, including data contained in external content files.

Step 2: Create a HTML page of the data view with sample data filled-in and with the desired formatting and positioning.

Step 3: Standardize formatting by creating a Cascading Style Sheet (.css) file and any dynamic behaviors by creating a script file (.js).

Step 4: Manually transform the HTML document into a valid XML document (also creating a valid XHTML document).

Step 5: Manipulate the XML document into an XSLT file, replacing sample data with the content from content files.

Step 6: Abstract the XSLT file so that the transformation will be valid from any relative location within the file hierarchy. This step requires knowledge of where an XML file is parsed, where the XSLT is processed, and where the HTML is rendered. How this is accomplished is described later in this chapter.

The task of creating a valid XML document from HTML in step 4 above is not automatic. WYSIWYG HTML editors such as Microsoft's FrontPage or Macromedia's Dreamweaver produce valid HTML, which doesn't necessarily translate into valid XHTML. The author used HTML-Kit by Chamio.com to assist in producing XHTML from HTML documents. That software uses the W3C's HTML-Tidy plug-in in the process of producing XHTML. As a converted document, the author found that further modifications were necessary to produce a valid XML document from HTML. For example, the attribute *selected* in the HTML <option> element is legitimate by itself; however, it was necessary to set the attribute to a value to make it valid XML (i.e. selected="true").

2. Data View File Dependencies

The robustness of XSLT allows for the conglomeration of numerous stylesheets and data files. For each data view developed, there is a minimum of three data sources being queried in preparing the HTML page for the user agent. While the initial trigger for the XSLT is a single XML file, other data sources are drawn into the processing via the <xsl:document> instruction.

Figure 8 illustrates a simple example of the dependencies between data sources. When viewing information about *people*, the user may navigate to other content associated with that particular *location*. The data necessary to provide the hyperlinks for the navigation is contained within the *Location.xml* file. Also, each *person* may be associated with a *resource*, and the association between a particular *resource* and *people* is via the reference identification (refId) attribute within a *person* content node. The XSLT processor dynamically associates *people* with a *resource* as the HTML page is being rendered.

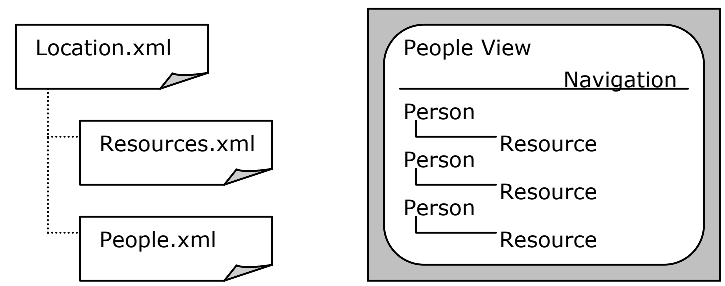


Figure 8. Data View Dependencies Example

The *main location index* and each *location* file are also dynamically associated. For each view of a particular *location*, the *main location index* is searched for the parent, siblings, and children of that *location*. These relatives are included in the HTML page rendered as part of the navigation feature of the system.

3. Relativity in XSLT and HTML Processing

Figure 9 demonstrates a simplified view of how the focus of relative URLs changes during the processing and subsequent rendering of HTML in a user agent. Keep in mind that both the XSLT processing and HTML rendering are accomplished client-side and not at the server.

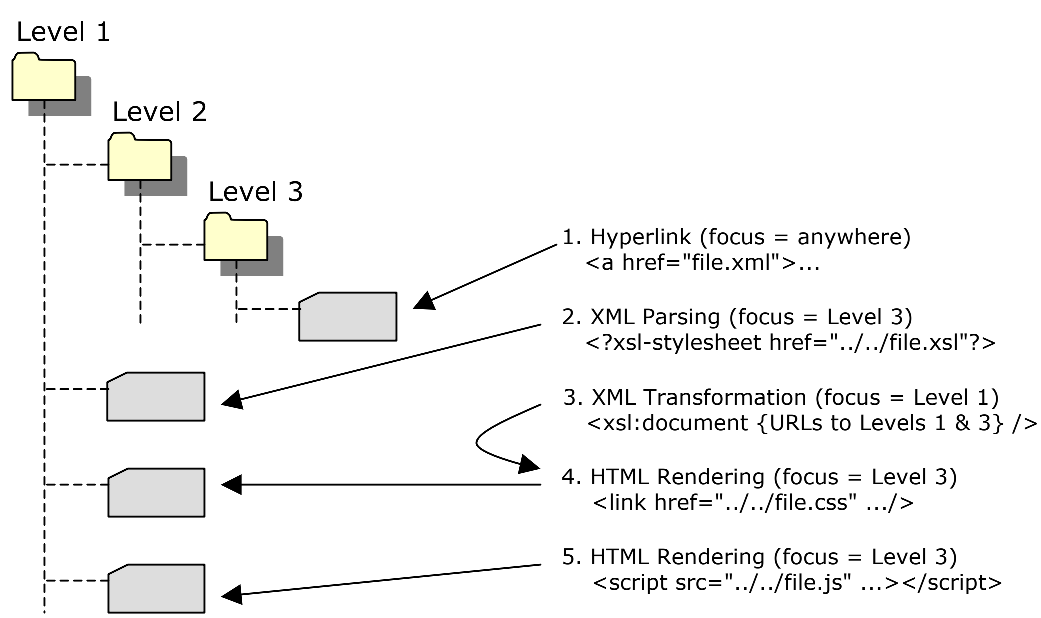


Figure 9. XML Content File To View Transformation

When a hyperlink redirects the user agent to an XML file, the server delivers the XML file unparsed. In the case of the developed system, the MSXML parser receives the XML file, parses the document, and discovers the XSLT processing instruction. Internet Explorer then requests the corresponding *.xsl* file from the server. Upon getting the XSLT file, focus shifts from the folder that contained the XML file, to the folder that contains the XSLT file. In order to get sibling files of the XML file (e.g. *Resources.xml* and *Location.xml* from the previous example), there needs to be information to "path" back to the original file. That information is derived from the *main location index* XML file with the knowledge of the corresponding location's *id* value (this is why there is a *parentId* attribute in all content files). The path to the *main location index* XML file is static and thus always known by the XSLT file.

The system is also designed so that the same Cascading Style Sheet (CSS) and JavaScript (JS) file is used by all content views. Even though the location of the CSS and JS files are statically relative to the XSLT file, the reference to these files is not used until processing is shifted from MSXML parser to Internet Explorer (a second shift of folder focus). In order to "path" to the *.css* and *.js* files, the XSLT engine must derive the relative URL path. The derivation is done using a sub-string of the XSL processing instruction <?xsl-stylesheet?> contained within the XML file that called the XSLT.

C. CONTENT NODE SCHEMAS

XML Schemas were developed for content nodes vice content files. Content files are simply a collection (or container) of content nodes. Individuals author content nodes, so the logical connection to error checking at the time of creation is tied to content nodes rather than their files. Thus, a content node can be validated against an XML Schema at time of entry before it is sent to the system for processing. Likewise, a content node can be validated upon receipt at the system before it is saved. The latter validation also allows the system to operate independently from the source of the data, thus allowing for third-party and web service interaction with the system as a potential for future enhancement.

D. DYNAMIC FORM ELEMENTS

1. Multiplicity of Data Elements

Relational databases suffer from having to reference associated data with a cardinality of many (N) to a related table. However, XML data files may have a multiplicity of elements greater than one embedded within them and still be able to validate the data with the use of an XML Schema. In our natural lives, the occurrence of the above example is frequent. In the developed system there are two notable examples. First, routes (or tracklines) that ships navigate are comprised of coordinate waypoints of latitudes and longitudes. Second, people often have multiple means of being phoned (e.g. business, fax, home, cellular phone, etc.). These examples require the design of a data entry system that is flexible to meet both needs, without any 'unnatural' restrictions. Creating dynamic form elements was an essential part of the development of the system.

2. Enabling Dynamic Form Elements

Dynamic form elements require two important features: a blank template of form elements (a group) to be dynamically created or deleted, and a mechanism to maintain reference to each group of elements. In order to maintain a template, two requirements are necessary. First, using the CSS feature of *style="DISPLAY: none"* attributes within HTML elements hide data from the user's view, but are still available for DOM HTML (or dynamic HTML) to manipulate them via user actions. Second, form elements outside of a <form> tag set are not part of the *forms* collection of DOM HTML. This means that a form processed or posted to another page does not contain information from elements outside of the referenced form.

The mechanism to keep reference to groups of form elements is through the use of *id* attributes within <div> tag sets. <div> tag sets organize form elements into groups which can be dynamically controlled within a form. The *id* attribute in the developed system consists of a time-stamp value pre-pended with a text character (due to HTML requirements). Actions associated with a group of elements (add/insert/delete) simply point to the parent element's *id* (within the <div> tag) as a reference to perform the action upon.

E. GENERALIZED ADD AND MODIFY PROCESS

1. Chosen and Alternative Design Patterns

Figure 10 illustrates a simplified view of how the developed system's data entry and modification work. A transmitter receives directions via a URL query string, which contains the desired action of the user. The transmitter creates a dynamic HTML form page in which the user inputs information. The HTML form data is then sent via an HTTP message using the SOAP format to the receiver. The receiver processes the data and replies with an appropriate message. Data validation is performed at both the HTML form page and the receiver.

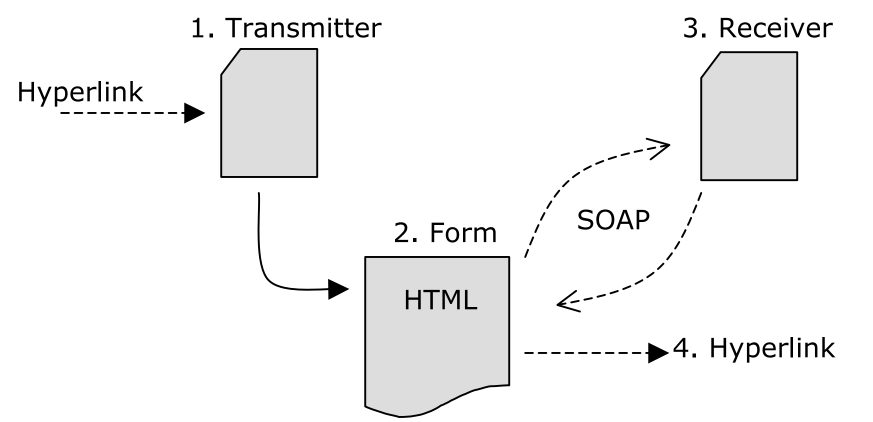


Figure 10. Data Entry And Modification Overview

The above design was chosen over the proof of concept system developed and one other design considered. Among the criteria of selection, three features were central to the application developed. First, data validation at the client using XML Schema is preferred over the use of validation via a scripting language. It is difficult to capture and implement the robustness of XML Schema data validation in JavaScript. Even more difficult is managing the changes between XML Schema and its associated JavaScript validation. The trade off in using XML Schema at the client is the terse and sometimes difficult to understand error reporting.

The second criterion was the manner in which HTML form data is posted to a server. Data from a form is collected and sent to a server as a series of name-value pairs. In doing so, there is a loss of identity in what type of form element contained the data,

and its use within the form. For the designed application, hidden input tag elements provide the ability to communicate 'overhead' information about the content of the form. Keeping associated data separate allows for a more efficient means to create content nodes from DOM HTML to DOM XML via iterative loop vice element-by-element. Once the form data is in a DOM XML object, the creation of a content node is accomplished by a simple XSLT.

Finally, the use of SOAP creates the potential for the system to be expanded as a receiver of information from many sources, independent of how the information is generated. In the current system, the HTML form page is not tightly coupled with the receiver of the information.

2. URL Query String Contents

In the prototype application, URL query strings are used to communicate the intent of the user to the system. Dynamic information exchanges via a URL query strings was explained in Chapter IV. When communicating to the *xkmsTransmitter.asp* file the following four query string arguments shall appear, otherwise an error is reported:

- nodeType=[Location | Route | Pier | Resource | *etc.*]
- nodeAction=[Add | Modify | Delete | Vote]
- nodeId={id value}
- refId={refId value}
- folderPath={folder path with '/' characters escaped}

One disadvantage of URL query strings is that HTTP *Get* operations (which use query strings) pose an unnecessary computer security risk. The author consciously choose to develop the application as described above in order to provide the ability to test and debug the system. Further implementations may need to re-address this issue.

3. Actions of the Transmitter

Figure 11 illustrates the general purpose of the *xkmsTransmitter.asp* file. The purpose of this file is to generate for the user agent the HTML form page for adding or modifying data to the developed system. The process begins with the receipt and processing of the URL query string as outlined in the previous paragraphs. The data from the query string is encoded into hidden <input> tags, which are later appended to the generated form. The author chose to keep the processing instructions with the input data to enable the potential receiver to act as an independent web service.

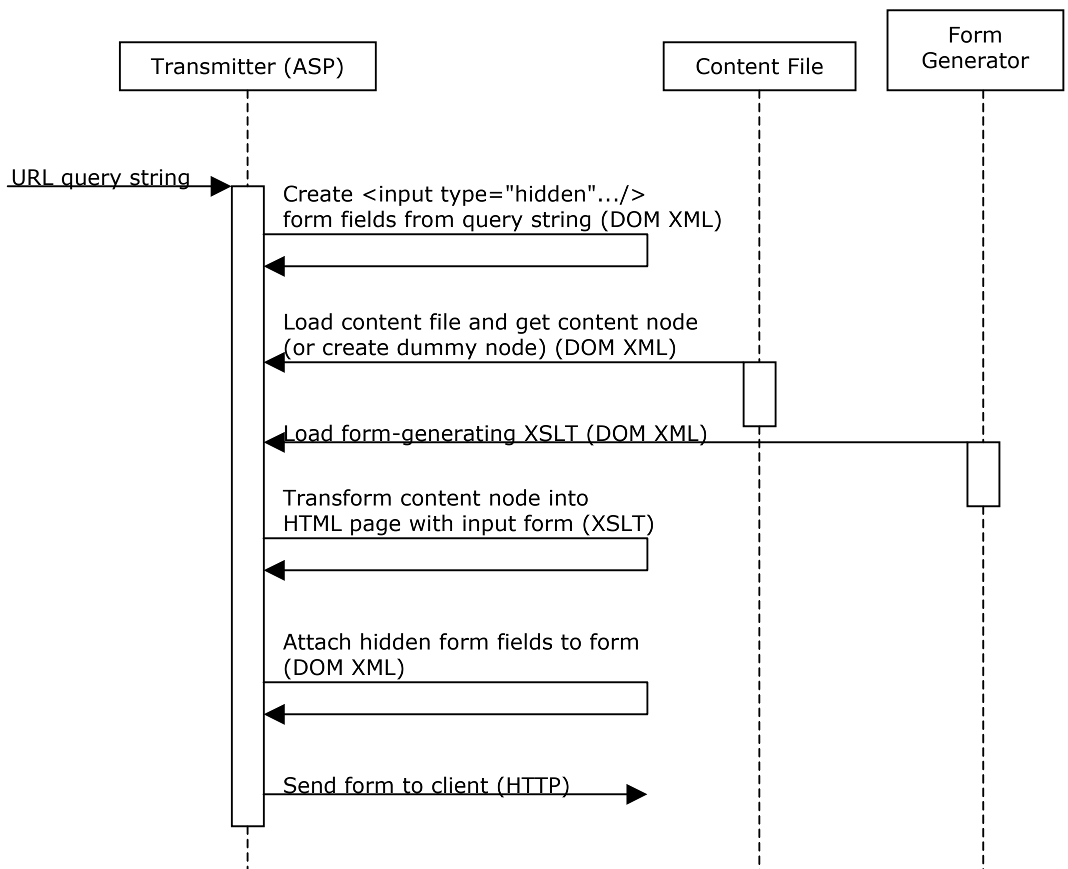


Figure 11. HTML Form Generation Actions

The mechanism to generate the HTML form is an XSLT. The XSLT acts upon a DOM XML document which is either a content node in the case of a modification of existing data, or a "dummy node" (i.e. an empty element) if the operation is to add data to the system. This design takes advantage of the fact that XSLT processors do not generate an exception if the data is not found within the target XML file.

The output of the XSLT is a complete HTML page. As mentioned above, the hidden input fields are appended to the <form> element and then served to the client. From a design perspective, the transmitter performs a one-way function: get instructions and produce a HTML form page. The only exception is minimal error handling to detect processing instructions in the URL query string that are incomplete, and to redirect the user if they are not logged into the system. If the user is not logged in, which is represented by a server session variable similar to the proof of concept system, then the form generated is a logon form. In this case, the user processing instructions are saved as described above. When the receiver of the from processes the logon data, it will automatically redirect the process back to the transmitter with the instructions originally sent by the user.

4. Actions of the HTML Form

Figure 18 illustrates the actions of the generated HTML form pages. The general purpose of the form is to collect data, transform the data into a content node, validate the content node, and then send the content node to the system for processing. This process begins when the user clicks on the *submit* button on the form. The submit button fires a JavaScript function vice the browser's default behavior of performing a HTTP Post to a server.

The first step in creating a content node is to gather the contents of the form elements. The collection of form data elements is represented by a linear array of items (which follows the top-to-bottom and left-to-right page flow). That collection is iterated through using DOM HTML and the output is a linear list of elements in two DOM XML documents (one for user entered data and one for system data). The root element of the first DOM XML document takes the name of the HTML form and its child elements take the name of the form data elements. There is nearly exact duplicity between form

<input> element names and the corresponding XML they represent. The DOM XML document is then transformed into a content node using an XSLT. During the transformation, default values are entered for *id*, *refId*, and date-time stamp attributes, which are later given permanent values by the receiver. The second DOM XML document is a collection of hidden form elements. Again, these hidden form elements represent the processing instructions to the *xkmsReceiver.asp* file.

The linear nature of a HTML form weakens the ability to create nested data sets in XML. Several content nodes have three nested levels of data (e.g. a *latitude* is nested within a *waypoint*, which is nested within a collection of *waypoints*, which is contained in a content node). The XSLT that transforms the linear XML representation of form data into a content node uses triggers to set up the inner nested XML tags. This may create potential problems. For instance, a phone's *location* form element triggers the transformation of a *phone* element. If no *number* is present directly following the *location*, nothing is lost. However, if a user enters a *number* with no corresponding *location*, then the entered *location* will be lost in the processing. While the XML Schema can detect such anomalies, further research is necessary for finding a suitable remedy.

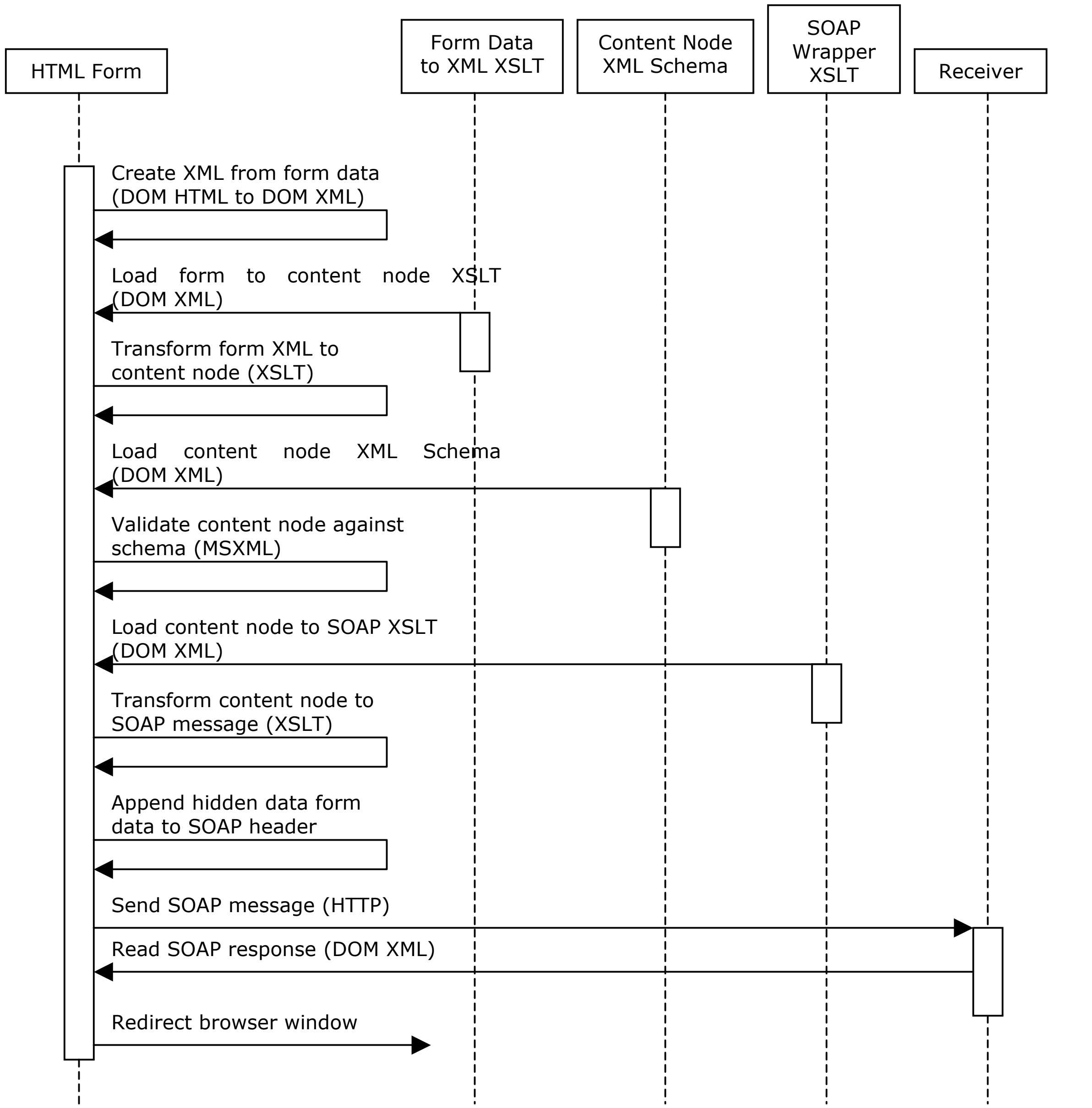


Figure 12. HTML Form Actions

Once the content node is created, its corresponding XML Schema is loaded. The name of the file to be loaded is again derived by the *name* attribute of the HTML form used. The schema is then compared to the content node, and any validation errors are reported to the user. This process has a distinct advantage over the typical server-side error detection of data in that the form's data is processed without transferring the user to another HTML page. The user benefits because they have the ability to correct the data without re-entering it. In fact, the browser will return the data to its original state during

a modification if the 'reset' button is clicked, vice clearing the data completely from the input fields (as observed using Microsoft's Internet Explorer).

If the content node is valid, a second XSLT file is loaded to act as a wrapper function in creating a SOAP message. The target of the XSLT is the content node, which simply places the content node inside the body of the SOAP message. The DOM XML document that contains the hidden form data is placed into a header block of the message. The SOAP message is then sent to the *xkmsReceiver.asp* file in a HTTP Post transmission.

Two outcomes are possible from the SOAP message that is returned by the receiver. If a data error occurs (as indicated by a SOAP Fault message), the user is notified and the focus of the browser remains on the form. If another type of error occurs, then the browser is redirected to a separate *error.asp* page to provide the necessary feedback to the user. Finally, a successful entry into the system will result in the user agent's window being redirected to viewing the new data (using DOM HTML).

5. Actions of the Receiver

Figure 13 illustrates the general actions of the *xkmsRecever.asp* file. The actions of the receiver are the most complicated aspect of the system. The receiver's role is to receive, validate, process, and store new and modified information in the system (a controller pattern). The receiver's inputs and outputs are SOAP messages. Error reporting is also handled via SOAP messages.

The receiver's processing begins by extracting the first and only header in the received SOAP message. Within the header are the processing instructions that were described in paragraph E.2. of this chapter. The instructions become page-level variables within the ASP. As the processing continues, separate functions are called to handle unique types of content (logon data, location, votes, and all others).

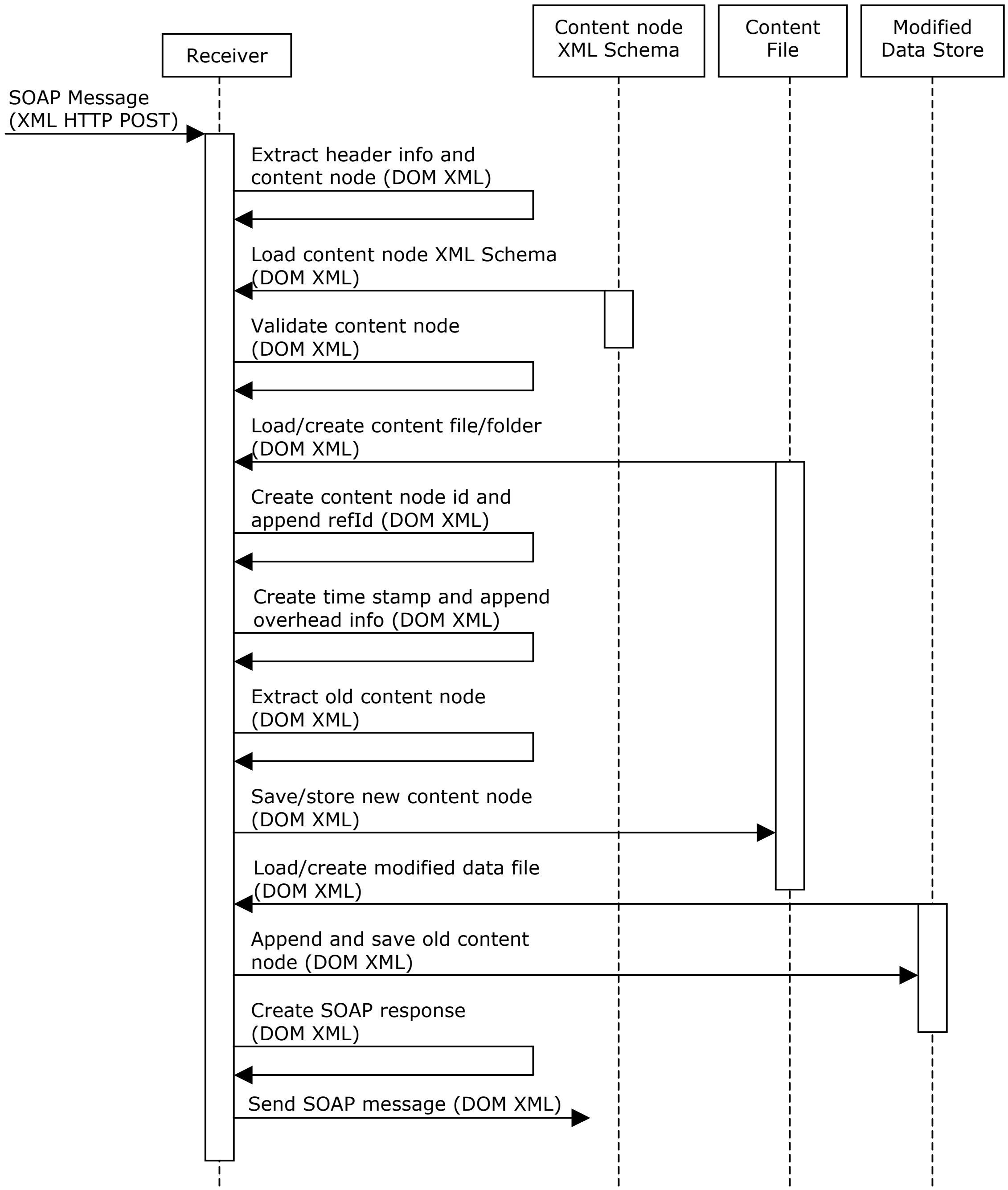


Figure 13. Receiver Actions

The next phase in the processing is dependent upon the actions to be taken on the data. For additions and modifications, the incoming content node (which is the SOAP message body) is validated against its corresponding XML Schema. If an existing content file exists, then it is loaded into memory, otherwise it is created. Valid content nodes are then modified to contain an accurate identification (id), reference identification (refId), Internet Protocol source address (ipAddr), and time-stamp (modifiedDateTime) attributes.

In completing a transaction, up to three XML files may be involved: the target data store, the modified data store, and the index that refers to the previous two files. The modified data store is a collection of content nodes (by content type) that have either been modified or deleted. Table 5 details the files that may be manipulated during the processing of data.

	Main Index Location.xml	Location.xml	(General) *Content*.xml	modified_ *Content*.xml
Add Location	X	X		
Modify Location	X[1]	X		X
Delete Location	X	X		X
Add Content		X[2]	X	
Modify Content		X	X	X[2]
Delete Content			X	X
Vote			X	
Logon				

[1] - If the title or folder name for the location changes

[2] - For the first addition of an content node

Table 5. Files Modified During Content Processing

There are two important facets regarding the above table. First, the main index *Location.xml* file and the sub-*Location.xml* files both act as indexes in the system. The *main index location* indexes all other locations and all the other locations index the content that pertain to that area. Second, nothing in the system gets permanently deleted. This is a design feature intended to act as an archive and audit trail to the modifications made to the system. The logon data is appended to any content node that is either

modified or deleted. Any implementation of this system should consider the need for being able to "undo" data modifications and the long term storage of archived information.

The remainder of the processing is a matter of adding or replacing content nodes to the content or modified content stores and adjusting the indexes to data as necessary. If the transaction completes successfully, the response to the client is a SOAP message which contains a URL to the new or modified data, or to the parent location when a deletion occurs.

F. UNIQUE TRANSMITTER AND RECEIVER PROCESSES

Some of the processes described in the previous section have unique sub-processes to handle special types of content. For example, the *location* content node and file requires handling of both XML files and file system folders. Managing file folders and their attributes provided unique challenges, including creation, modifying, and naming conflict resolution. Managing the indexes also required special programming instruction.

A separate sub-process also handled voting on the quality of the existing content. This action bypassed the normal logon procedures and did not require any interaction with *location* or modified content files. At the *receiver*, voting also required a unique process since the data already existed and remained in the same content node.

Lastly, the creation of a confirmation HTML page for deleting a content node of any content type was abstracted to one XSLT at the *transmitter*. While this required special coding, the *receiver* did not. Processing the deletions of a content node was handled by the unique behavior of DOM XML explained in Chapter IV. That is, appending a content node without cloning it acts as an extraction. As a side effect, the deletion process is very similar to the modification of data.

G. XML PROCESSING CONSIDERATIONS

1. Client-Side Loading

In the context of browsing data, the developed system can be viewed as a simple file server where all of the processing is performed at the client. Specifically, all XML parsing and transformations, formatting via CSS, and dynamic behaviors via JavaScript

and DOM HTML are performed using the client's CPU and resources. While this design places some burden on systems administers to ensure the proper software is installed on the client, it lightens the load of the server so that it may handle a greater volume of data requests. More importantly, it fulfills the requirement for the system to be deployable in a connectionless non-server environment.

2. Server-Side Loading

In addition to the file serving as described in the previous paragraph, nearly all other server-side processing occurs from two ASP pages (*xkmsTransmitter.asp* and *xkmsReceiver.asp*). Having only two processing pages greatly simplifies the design of the system, but may have an impact on server loading. Server-side issues have not been researched and should be considered before full implementation.

H. LIMITATIONS OF EXISTING SYSTEM

1. Data Normalization

Unlike relational databases with its normalization forms, there currently does not exist a proven methodology for constructing an XML database. Some aspects of XML data storage cannot use the rules of relational databases. "One difference between native XML databases and relational databases is that XML supports multi-valued properties while (most) relational databases do not.[3]" For example, an XML data file may have an array-like list of data elements, which can be validated against a schema. Duplicated fields within a table do not conform to the first normal form of relational databases. Those said, the other principles of normalization are still relevant and were considered in designing the data schemas.

As a result of numerous design changes, the duplication of system data was kept to a minimum. The *main index location* file is an XML repository of *location id*s, and the location type, folder name, and title name for the location. The duplicity was chosen to improve the speed of the application.

2. Referential Integrity

"Referential integrity refers to the validity of pointers to related data and is a necessary part of maintaining a consistent database state.[3]" In the developed system there exists five instances of 'pointers' that relate the data, one of which is potentially harmful.

- The *main index location* XML file contains the *id* values of each *location* within the system.
- Each *location* file contains a list of content files that exist for that location.
- Each *comment* node contains the *id* value for its content.
- Each *person* node may contain an *id* value for its associated *resource* or *activity*.
- Each content file contains the *id* value of the parent *location* node.

The last pointer creates a bi-directional reference situation. *Locations* store information about its content files, and content files store information about its *location*. The need for the latter reference in the content file has to due with the need for creating a navigation window within each content file view. During the XSLT processing of a content file view, the location's *id* value is used to create a URL path back to the folder (using the *main index location*) to include the related data from the *location* and other sibling content files. The need for this pointer would be moot if each *location's* name were guaranteed to be unique (which is a key by definition).

3. Searches (Data Queries)

Due to time limitations, no capable search function was developed for the system. Some query-like functions were built into some of the view XSLTs, which are shown as hyperlinks to related data held in other areas of the system. Research into the utility XPath and XSLT as query tools could be conducted on the developed system. The system would also be a potential source for research in XQuery as APIs become available.

The lack of specific search tools is not in itself a roadblock towards implementing a system such as this. Because the file contents are in plain text and they are stored within the visibility of the server and its hyperlinks, it is inherently in open view of both operating systems file search utilities and Internet web crawler software. And, with the

advancement of web searching software, there may exist legitimate reasons for relying on such tools for searching the system.

4. Record Locking

The developed system has the potential for very high concurrency (i.e. the number of simultaneous users). However, it has no record locking mechanism. The system behaves in a way that only the last modification is retained, all others are collected in a modification collection file. This is not a desirable feature and should be included in any further research on this subject.

5. System Integration

The developed system does not include the capability to interface with any of the common industry database connection protocols (ODBC, OLE DB, JDBC, etc.). This condition is further worsened by the inability to use SQL commands and the lack of query tools. These limitations are relevant as concerns for interoperability only if there are other systems that have or foresee a need to interoperate with the developed system.

Web services provide one potential solution to the interoperability problem. Since the data is stored and represented in a non-proprietary format, web services could be built to provide data from the system using WSDL and SOAP messaging. The tools required for accessing the data are not bound by the other proprietary technologies used in creating the existing system. SOAP messaging has already been implemented as a mechanism to logon, add, modify, and delete data, and could easily be extended to provide specific content from the system.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. XML DATABASE SELECTION AND FUTURE TECHNOLOGIES

Having chosen a particular development path for the proof of concept application, it is prudent to consider the alternatives that were either decided against or overlooked. This chapter is divided into two sections. The first section is a description of the selection path considering the alternative choices. Its intent is to put into perspective the growing availability of database tools and technologies. Having selected a particular path, the second section is devoted to analyzing the future of XML data and presentation related technologies that could influence the application developed.

A. DATABASE SELECTION STRATEGY

It can be argued that the selected storage strategy lacks the robustness of existing storage technologies, which weakens the performance and reliability of the overall system. A contrary view can be argued with Clayton Christensen's definition of a disruptive technology and how it adds value.

> [A disruptive technology is a technology or innovation] that results in worse product performance, at least in the near-term...[and] brings to the market a very different value proposition than had been available previously...Products that are based on disruptive technologies are typically cheaper, simpler, smaller, and, frequently, more convenient to use. [However, they] underperform established products in mainstream markets[5].

The developed application is just such a product. Its intent is to serve a group that has been underserved with insufficient means to share, store, and use knowledge. And though its solution currently lacks the utility of other related techniques, the development of supporting technologies offer great potential in performance rewards in the near future.

1. Selecting the Data Storage Representation

The alternatives to selecting an XML storage system are relational and objected oriented databases. Relational database systems are ubiquitous, and SQL is a very capable querying mechanism for accessing data. However, operating a relational database generally requires lock-in to a particular vendor due to proprietary extensions to the query language. Object oriented databases suffer from the same issue but to a larger

extent (primarily due to the failure to adopt querying standards such as SQL-3). Portability of systems to environments without network connections would require product licenses for each user of the database and additional training for using the application interface. Though Microsoft Access is installed at every client within the Coast Guard, its inability to handle large data sets and high levels of concurrency makes it unsuitable as a corporate wide server-side database system.

Not selecting a well-established data system presents many challenges in selecting an alternative. Relational databases can store data using compression techniques, have highly optimized indexing systems, include security features, implement a variety of concurrency controls, and are often packaged with an integrated application development environment. Developing these mechanisms from 'scratch' is possible, but comparatively expensive (cost and time) to existing off-the-self software.

The selection of the storage system also has to take into account the natural environment of the data. The popularity of using XML to communicate e-commerce transactions is growing rapidly. Documents that exist naturally as XML require a mapping mechanism to transform the contents of the XML into a table for relational databases or an object for an object-oriented database. The binding process creates overhead for the data system, which if not recorded precisely can lead to a loss of structure in the original XML content. This behavior is called round trip engineering, an issue of ensuring the storage and subsequent retrieval of XML results in equivalent documents[3].

For data that currently exists, nearly all of the commercial database applications have been enabled to export and import XML data. The general format of the exported data is an XML document with a flattened structure similar to a HTML table layout with additional meta tags to clearly specify the source. From this perspective, XML can be viewed a common denominator of interoperable data sharing rather than a necessary data storage system. For existing data, XML can be an enabler rather than a barrier (assuming a common data typing schema is used, such as XML Schema).

Lastly, selection of the database storage system must be a strategic decision. If there are long-standing processes that provide and consume data using one particular

technology, it is a logical to continue using that type of technology. If the process is new (like in this case) there must be a forward-looking view of how the process will relate to the strategic vision of data storage and interoperability. Additionally, there are inherent risks with using an immature technology, which has the potential to be obsolesced or not adopted in the near term future.

2. Selecting the XML Storage System Type

There are benefits to storing data as XML. XML tag sets are self-describing and human readable[3]. Unicode is the text standard for XML, and thus standard across nearly every computing system. Data from structured XML documents can be retrieved faster than from relational database systems. Normalized data broken into separate but related tables require computationally expensive join functions to extract the data, whereas related data can be structured and stored with the same XML document[3]. XML can be semantically organized into tree or graph structures. XML is particularly suited for document-centric or semi-structured data.

Before selecting a XML database system, the data structure and data types need to be analyzed. Data stored in relational database systems are generally viewed as tables of data fields. This type of structure lends itself to data-centric documents where each field can be validated against a particular data construct. The contrary is document-centric content. This type of content can be thought of as containing human readable text. People's opinions, directions, and thoughts are often communicated using free flowing document-centric text. In practice, most data will be of some varying degree between data and document centric.

When selecting an XML database system there are three different alternatives: a hierarchical file structure (like the developed system), blobs within a relational database, or as native XML overlaid in a relational or object oriented database subsystem. A blob is an XML document stored as a large text field[3]. Upon entry, an XML database that stores blobs may create indexes to reference the XML content.

Before native XML databases become mainstream, there are serious challenges that need to be solved. First, XML document tag sets can cause documents to be very verbose. As an example, the main location index in the developed application can be

reduced to a sixth of its original size with compression. Second, current multi-document searching techniques can be very slow. The current situation prevents any such solution to be used in a highly competitive commercial sector. Third, a standardized XML query language has yet to be agreed upon. While it is likely that the W3C's efforts will prevail, it will take some time for a common set of tools to become ubiquitous.

3. Native XML Products

As of this writing, there are approximately thirty-five native XML database products[2]. One-quarter of them are open source projects, and their underlying data systems are a mix of proprietary implementations, relational, object oriented, and several miscellaneous types[2]. The common features of these systems include an API to perform data manipulations, a means to navigate the data, and possibly a query syntax using one of the developing standards (e.g. XQuery, XML-QL, or XML:DB).

Of particular interest is the research conducted by Nambiar, et al[27] on the performance of native XML database systems. Their research tested four different native XML databases, each having a different means for indexing, data representation (e.g. DOM model, document fragments, etc.), and data storage (e.g. file system, relational, and object oriented). Their data demonstrated that storing XML content as *.xml* files provided faster response times for search and navigation queries than documents broken apart and stored into optimized patterns[27]. As for selection of one type of native XML database application, it may be still too early to form judgment on the best design due to the rate at which standards and processors are being developed.

For the developed application the selection not to use a native XML database application was made primarily on avoiding the requirement of using a commercial, and potentially proprietary, product. An additional side effect of using a native XML database is the loss of control. XML documents entered into a system are 'digested' and stored as system dependent files. For the developed application, it was required that the original content remained visible to the browser software throughout its lifecycle.

B. ONGOING XML DEVELOPMENTS

In this section, the progressing futures of XML technologies are discussed. Each of them could have a positive impact on the system developed. All of these technologies add to the ever-expanding XML universe, both in size and complexity. Figure 14 illustrates some of the technical connections between these technologies. While the connections are incomplete, its intent is to show two features. First, XML Schema and XPath in their current and proposed states are at the heart of more complex languages that extend their capabilities. The observation of a 'web' of technologies is appropriate. Secondly, the stack relationship of the bottom three technologies builds outward as a notion between 'knowing where you are' and 'knowing exactly where you're going'. Examples are provided to demonstrate how these relationships exist.

1. XQuery (An XML Query Language) and XPath (Version 2.0)

XQuery and XPath 2.0 both became W3C Working Drafts on November 15, 2002[41, 46]. XQuery is an effort to standardize a methodology of querying XML data. The effort to create XQuery began in 1998. It is built as an extension of XPath version 2.0. Approximately 80% of the foundation for XQuery is XPath[17]. While XQuery has a large role in the development of XPath 2.0, improvements to the original Recommendation have come from standards that followed the original specification (e.g. XML Schema and XML Information Set) and to parallel ongoing improvements in other standards (e.g. XSLT 2.0).

Out of necessity to provide a highly reliable and repeatable querying language, XPath uses XML Schema data types and validation. Errors can be raised if incorrect data types are discovered during processing, and documents can be validated as part of a new command set[46]. Unlike XSLT version 1.0 variables, XQuery variables have enhanced capabilities within XPath 2.0, especially in newly defined programming statements for sequence, conditional, iterative, and quantified statements.

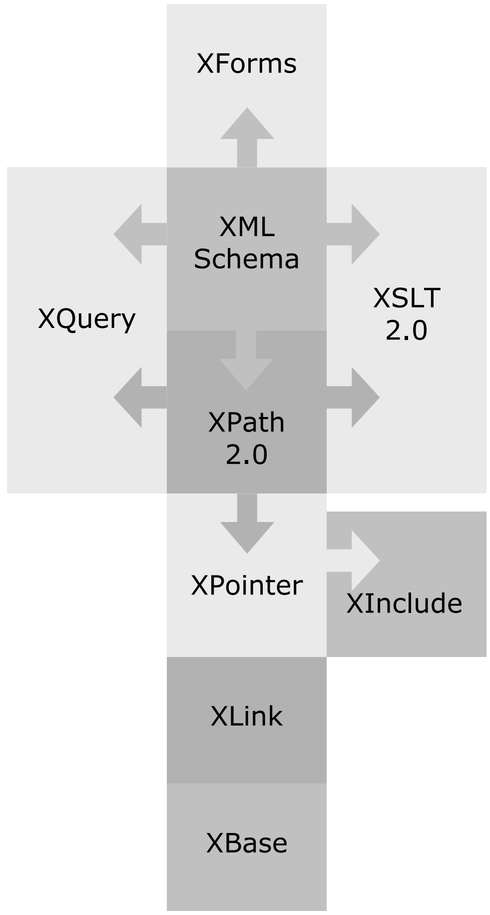


Figure 14. Relationships Between W3C XML Technologies Used

As mentioned, XQuery extends XPath's power by providing additional query-specific language commands, the capability to create user-defined functions, and ensuring the typing of data returned[17]. The definitive goals and requirements for the new XQuery language are published as documents that parallel the development of the language itself[46]. In addition, use cases and a data model description accompany the specification.

The initial submission for the XQuery language was based on an XML query language called Quilt, which in turn was derived from the Object Query Language (OQL) approach and the Structured Query Language (SQL) (Chamberlin, pg 598). Added to the development were the important facets of XPath and XML Schema. At the core of XQuery is the *for-let-where-result* (FLWR, pronounce 'flower') functionality, and

the *union*, *intersect*, and *except* statements[4]. With these and the other XPath tools, the XQuery language is a very capable. The acceptance of the XQuery language will be determined by the performance of processors that are developed.

The progress of XQuery towards becoming a Recommendation has significance. First, it represents an alternative to entity-relational and object-oriented database systems. Relational databases with SQL capabilities are entrenched in nearly every business. In contrast, object oriented databases have proven their superiority in some performance areas, but its weakness of standardization and interoperability has lead to a lack of acceptance. A W3C Recommendation of XQuery has the potential for commercial systems to be highly interoperable. Second, XQuery represents the notion that the W3C intends to be an XML standards body, not just a web standards body. It can be argued that XQuery has more to due with creating a non-proprietary and interoperable data language, and less to due with creating a web-based technology.

2. XML:DB Initiative

The XML:DB Initiative is a non-profit collaborative community with the goal of developing a standard for accessing data in an XML database. The XML:DB community is represented by twenty-three companies (mostly from the technology sector). In their view, "XML databases, have much greater applicability than just the World Wide Web and it is for this reason that [they] felt it was the appropriate time to form a new organization chartered with the development of XML database specific specifications[44]." Initial work by the XML:DB has been incorporated into Sun's Xindice XML database and Software AG's Tamino database system[44]. Currently there is insufficient data available to compare or contrast the performance differences between the XML:DB initiative and XQuery.

3. XForms

XForms became a W3C Candidate Recommendation on November 12, 2002[35].

> XForms is an XML application that represents the next generation of forms for the Web. By splitting traditional XHTML forms into three parts—XForms model, instance data, and user interface—it separates presentation from content, allows reuse, gives strong typing—reducing the number of round-trips to the server, as well as offering device independence and a reduced need for scripting[35].

HTML forms have been in use for ten years, twice as long as XML. Their original utility is not complementary to XML information sets. As discussed in Chapter V, HTML form-data to XML transformations are inherently lossy due to the linear nature of form fields. XForms is a robust solution to HTML form's weaknesses.

At its inception, XForms was designed to be a replacement of HTML forms. It will become a modular component of XHTML 2.0 (a complete re-write of HTML)[35]. The modularity of XForms will provide its potential reuse by other computing communities beyond XHTML. XForms offer the following benefits:

- **Strong typing** - using XML Schema standards for data typing, validation of form fields can be accomplished at the client, prior to data being sent to a web server.
- **Flexibility** - with separation of components and high level description of behaviors, XForms could be adaptable to many different browser environments (computer screen, PDA, television, etc.)
- **Less scripting** - the previous two features lessen the need for specialized scripting code for the HTML output (i.e. less JavaScript and VBScript).

The authoring of an XForms document is through a process of describing the data constructs (including data types), the presentation (e.g. XHTML), and binding the components together in a model[35].

The efficiency of the developed application could greatly increase with the use of XForms. The rendering of the form within the browser, decoupling the data fields from the form, and validating the data at the client are currently done with a combination of JavaScript and XSLT. An XForms processor (embedded within an XHTML browser) would automate most of work and enable a generic environment to create forms.

Implementation of a validated XForms processor, and subsequent development of a stable integrated development environment (IDE) will be difficult. The XForms specification includes over seventy new methods in its application programming interface (API), use of all existing XML Schema types and four extended ones, and use of XPath. While approximately two-dozen open source projects and small companies have developed incomplete solutions, no major vendor in the technology sector has implemented an XForms processor[35].

4. XML Base

XML Base became a W3C Recommendation on June 27, 2001[37]. The purpose of XML Base is to describe a standard XML attribute to achieve functionality similar to that of HTML's BASE element. The *xml:base* attribute sets a Uniform Resource Indicator (URI) location for all other URIs to be referenced. The base URI is completely independent of the URI of the source document.

Figure 15 demonstrates the use of the XML Base attribute. Note that XML Base uses the XML namespace. The use of the XML namespace functionally alters the baseline of the XML language. The processing of the *xml:base* attribute cannot be assumed, as this change to the language occurred after the latest edition of XML.

The following XML document containing an XML Base:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head/>
  <body xml:base="http://cgweb.uscg.mil/xkms">
    <a
      href="World/Location.xml"
      alt="Go to Location Index"
      target="_blank">
      Link to Main Index</a>
  </body>
</html>
```

Would resolve the hyperlink "Link to Main Page" to the following URI:

http: //cgweb.uscg.mil/xkms/World/Location.xml

Figure 15. XML Base Example

The use of XML Base would provide no added benefit to the proof of concept application. However, XML Base is important for two reasons. First, XML Base is an important building block for the W3C recommendations that follow the XML Base recommendation, specifically XLink, XPointer, XPath version 2.0, and XQuery. These recommendations either utilize or extend the utility of the *xml:base* attribute. Second, if the design requirements for the developed application change to restrict the deployment to specific locations (i.e. specific URL or file drive and path location), the difficulty of deriving file locations would be greatly reduced.

5. XLink (XML Linking Language)

XLink became a W3C Recommendation on June 27, 2001[39]. XLink is mature version of HTML's hyperlink <a> element (i.e. anchor tag) or <img> element (i.e. image tag). HTML links can only provide a one-way path, they can only point to one item, and they can only be described by one attribute. The background of XLink comes from hypermedia, where the connections between content can be expressed in a multitude of ways.

Figure 16 demonstrates the use of a *simple* type XLink. The processing of this example produces the exact same results as Figure 15. This XLink extends the previous example by associating the role of the link with a URI namespace and controlling the timing of the link with an *actuate* attribute. More advanced XLinks describe targets that exist within the same document or externally via URI, user readable labels for complex links, and the arc descriptions between the targets. Of approximately one-dozen implementations of XLink, Mozilla and the W3C's Amaya browsers are the two most notable tools (Mozilla only supports simple links).

The following XML document containing an XLink:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head/>
  <body xml:base="http://cgweb.uscg.mil/xkms">
    <MainIndex xmlns:xlink="http://www.w3c.org/1999/xlink"
      xlink:type="simple"
      xlink:href="World/Location.xml"
      xlink:role="http://cgweb.uscg.mil/xkms/Location"
      xlink:title="Go to Location Index"
      xlink:show="new"
      xlink:actuate="onRequest">
      Link to Main Index</MainIndex>
  </body>
</html>
```

Would resolve the hyperlink "Link to Main Page" to the following URI:

http: //cgweb.uscg.mil/xkms/World/Location.xml

Figure 16. XLink Example

The links formed by XLink can point to content internally within a document or externally to another resource. Links can describe both the source and the destination to permit circular operations. Bidirectional links are described as arcs, similar to

mathematical graph theory. Links can also be abstracted and stored in files to permit pointing to resources indirectly. Similar to DOM HTML events, XLink elements can describe the manner of how they behave and what actions from the user invoke the link[39]. Finally, links can also be described to provide greater meaning about their role and usage in the source document.

XLink is an appropriate technology where the contents of XML document and file collections are related. Therefore, any application where the use of foreign keys is used to describe the association between data sets is a potential use for XLink. In the developed application, the relationships between parent, sibling, and child locations, the relationships between a location and the various forms of content about that location, and the relationships between content types (e.g. resources to people and comments) can be described with XLink. However, use of XLink may not be suitable when the link references change frequently. Management of highly dynamic links may be better suited to runtime references created through XPath processing.

6. XPointer (XML Pointer Language)

XPointer became a W3C Proposed Recommendation on November 16, 2002[45]. The purpose of XPointer is to address internal fragments of an XML document from a URI. XPointer is akin to HTML URL references that include a pound symbol and identification attribute pair (e.g. http://www.page.com#paragraph1). However, XPointer includes and extends the power of XPath, which creates a power mechanism to ‘query’ an XML document externally without having to process it with an XSLT.

XPointer has been under development since mid-1997, a lengthy period compared to other technologies developed by the W3C. Having started out as part of the XLink development project, it has since grown and then been broken apart into four separate parts (the overall framework plus three unique functions: element( ) scheme, xmlns( ) scheme, and xpointer( ) scheme[45]). Although not fully recommended, its influence is already woven into other XML technologies, including XLink, XInclude, and DOM XML (Level 2). Another important facet of XPointer is its connection to HTTP, a standard created by the Internet Engineering Task Force (IETF). The implementation of XPointer directly influences the URI addressing schema developed by the IETF. A

related standard is being developed within the IETF to complement the usage of XPointer commands appended to URIs[40].

From the perspective of the proof of concept application, there is great potential for XPointer. The ability to selectively query an XML document and have the results processed by a XSLT would greatly improve the user's experience with the data. Currently, XSLT processing cannot receive user inputs. Work-around solutions include programmatically performing the XSLT operations iteratively at the client, or writing 'search' parameters to an XML document that is consumed by the XSLT (using the *document* function).

Figure 17 is an example of using XPointer within an XML document that has an *xml:base* attribute and XLink. XPointer adds to the power of XLink with the ability to navigate and select the data from the target document with the power of XPath functionality. As of this writing, their does not exists a usable XPointer browser.

The following XML document containing an XPointer:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head/>
  <body xml:base="http://cgweb.uscg.mil/xkms">
    <MainIndex xmlns:xlink="http://www.w3c.org/1999/xlink"
      ...
      xlink:role="http://cgweb.uscg.mil/xkms/Location#xpointer(//Location[@id='8675309'])"
      ...
      Link to Main Index</MainIndex>
  </body>
</html>
```

Would resolve the hyperlink "Link to Main Page" to the following URI:

```
http: //cgweb.uscg.mil/xkms/World/Location.xml#xpointer(id='8675309')
```

However, only the Location element with the matching id value would be return (which may be further transformed into HTML with a stylesheet reference).

Figure 17. XPointer Example

7. XInclude (XML Inclusions)

XInclude is a W3C Candidate recommendation as of September 17, 2002[38]. XML Inclusions is a complementary technology to the external entities defined in the original XML Recommendation[37]. However, XInclude is arguably more robust. An XML Inclusion is a call from a valid XML document for an inline replacement of an external resource. The external resources may be parsed XML information sets or other text-based media types. A URI identifies the source (or call to) the included resource. The URI may be extended with XPointer concepts to return a subset of an external XML document. If the included resource cannot be found or a parsing error occurs, an optional fallback option provides for alternative results.

The idea of inclusions is nothing new. Most programming languages have them, and other XML languages use them (e.g. XML, XSL/T, XSD). The power of XInclude comes from its ability to directly reference a resource, recover from some types of errors, and use the power of other XML languages (XPointer/XPath).

The utility of XLink described previously and XInclude have similar, but not overlapping use. XLink elements can be used to imbed content within the output to a browser (e.g. graphical images); however, there is no XML related processing that takes place. XInclude references can be parsed and included in an XML document for further processing.

The potential benefits of a XInclude processor in the proof of concept application could be significant. For example, the current situation requires the XSLT processor to test for the presence and load documents repetitively (e.g. testing for the presence of a *People.xml* file before searching for associations with a *Resources.xml* file contents for each resource content node). With XInclude, that type of processing could be accomplished more efficiently and prior to the transformation into a HTML document.

Figure 18 demonstrates a XInclude example. The XInclude processor is triggered by the use of the *include* element from the XInclude namespace. The URI of the target document includes an XPointer expression that calls upon the XPath processor, which subsequently returns only those personnel associated with the particular resource. If an

error occurs in parsing the document returned by the *include* statement, the result is an empty *Person* element.

The following XML document containing an XPointer:

```
<?xml version="1.0" encoding="UTF-8"?>
<Resources>
  <Resource id="123">
    <Title>USCG Station Monterey</Title>
    <Personnel>
      <xi:include
        xmlns:xi="http://www.w3.org/2001/XInclude"
        href="Poeple.xml#xpointer(/People/Person[@refId='../@id'])"
        parse="xml">
        <xi:fallback>
          <Person/>
        </xi:fallback>
      </xi:include>
    </Personnel>
  </Resource>
</Resources>
```

Would return the *Resources* document with all of the *Person* elements from the *People.xml* file with a *refId* matching the same *id* value.

Figure 18. XInclude Example

VII. CONCLUSIONS AND RECOMMENDATIONS

A. RESEARCH RESULTS

1. Objectives Accomplished

The prototype application is a collaborative XML-based knowledge management system that is transparent, deployable, and extensible using ubiquitous technologies and accepted standards. The primary research question has been answered with an operable system. In comparison to the available relational and object-relational database systems the developed application appears inferior in performance and capability. However, it satisfies very specific requirements for a set of users which cannot be achieved effectively with existing tools, including native XML database systems. The utility and power of advancing technologies within this field of study may eventually provide solutions to the current weaknesses.

Most of the effort in developing the application (50-60%) was spent in exploring potential designs, learning about the technologies used, and creating solutions to challenges faced. Having developed a unique process, the effort required to reproduce the work would require an estimated one-tenth of the original. The output of the research resulted in the following system files:

- 12 XML files - templates(10) for each content node (including votes and overhead), a blank SOAP message envelope, and an XML file describing the possible content types.
- 12 XML Schema files for each possible content type and one generic schema for the deletion process.
- 12 HTML form data to content node XSLTs.
- 12 XSLTs to create HTML forms from a content node.
- 9 XSLTs to create views of the data.
- 5 ASP pages (transmitter, receiver, media file upload selector, media file processor, and an error reporter).
- 2 JavaScript files (one for use by the HTML forms and one for the data views).
- 1 Cascading Style Sheet (used by every form and data view).

In addition to the system files, approximately 5,300 content files were created. The main location index file was initially created by hand coding the organization of the world into recognizable sub areas as described in Chapter II. Sample data for Monterey, California was also hand typed. A semi-automated process was developed for creating sample location data using a publicly available World Port Index (WPI) database. The WPI was contained in an Access database. A Visual Basic for Applications (VBA) script was created to read a query record set, transform the data into an XML location content node, and transmit the data using SOAP to the application's receiver. The receiver acted as a web service by processing the data into the system without having any knowledge about the creator of the data (location, language, operating system, etc.).

2. Performance

Only one measure of performance was recorded for the developed application, the time required to download and render the main index view. The main index *.xml* file is the largest data file in the system. With 5,225 locations, the file size is 486KBytes. This number of location represents a worst-case scenario (as there are only 4,679 locations in the WPI). Times recorded to download and process this file were:

- 100Mbps Ethernet (non-working hours) - < 3 seconds
- 100Mbps Ethernet (normal working hours) - 5-6 seconds
- 33.6Kbps modem - 39 seconds
- 28.8Kbps modem - 50 seconds

The above performance data reflects a non-optimized solution. No effort in the design process was made to reduce the size of the main index file or improve its processing efficiency. Significant gains could be achieved if the design were changed. For example, reducing the length of the element and attribute names can reduce the file size by 35%.

Further development efforts on this topic should include a clearly defined set of tests and performance targets. Of particular importance will be the ability to search for specific text, range of dated material, and content by particular data category (e.g., ports visited by icebreakers). As a qualitative test, the main index page was 'enhanced' with a search function (which searched the rendered HTML and not the XML data). That

technique proved disastrous, as it required over 30 seconds to complete the search (using 5,225 locations). An XPointer alternative should reduce the processing time to less than a few seconds.

The view and navigate portions of the application use non-proprietary technologies and are also free of proprietary extensions. An observation was made to test this by viewing the data with the Mozilla browser (an open source browser). While the XSLT transformations and JavaScript worked correctly, the Cascading Style Sheet behaviors did not. Mozilla's current inability to correctly display CSS content is well documented.

B. RECOMMENDATIONS

1. Implementation

The current application is a prototype, not a production ready system. Numerous programming anomalies exist. However, the difference between its current status and a corporate-wide solution is one or two development iterations. Having proved the concept can work and identifying the remaining challenges, the work required to redesign the requirements with user feedback, develop the application, and test the solution would require approximately four months. That work should be done in-house by the Coast Guard's own operations systems command. Failure to pursue the development of the application will result in the status quo of continuous re-creation and loss of knowledge.

The application was developed for a specific domain with one instance of knowledge (i.e., ship drivers and operational decision makers). It can be argued that the bandwidth available to ships at sea will not appreciably change in the near term future. With that assumption, the features of the developed application can be extended to other communities of knowledge in similar circumstances.

2. Further Research

Many areas for further research regarding the application were identified in the previous chapters. Those topics need to be addressed before the application can be implemented. However, the current application can also be used as a baseline for implementing a similar product in either a Java Server Page or ASP.Net environment. The Java route would increase the options available to implementing the server-side

portion of the system and the availability of newer and more capable XML processors (since most open source processors are written in Java). The transformation of the existing ASP pages (written in JavaScript) into an Apache Tomcat environment would no doubt increase the reliability of the system. The same improvements in performance may be possible in an ASP.Net solution, where server-side code is compiled and more thoroughly debugged than the run-time interpreted ASPs.

Exploration and testing of native XML databases is another area requiring further research. As this field matures, more robust alternatives will become available and begin to challenge the need for relational or object oriented databases. A solution may exist to fulfill the same requirements from an off the shelf product (open source or commercial). Investment in a new technology will require comparative performance testing, security verifications, and thorough risk assessments.

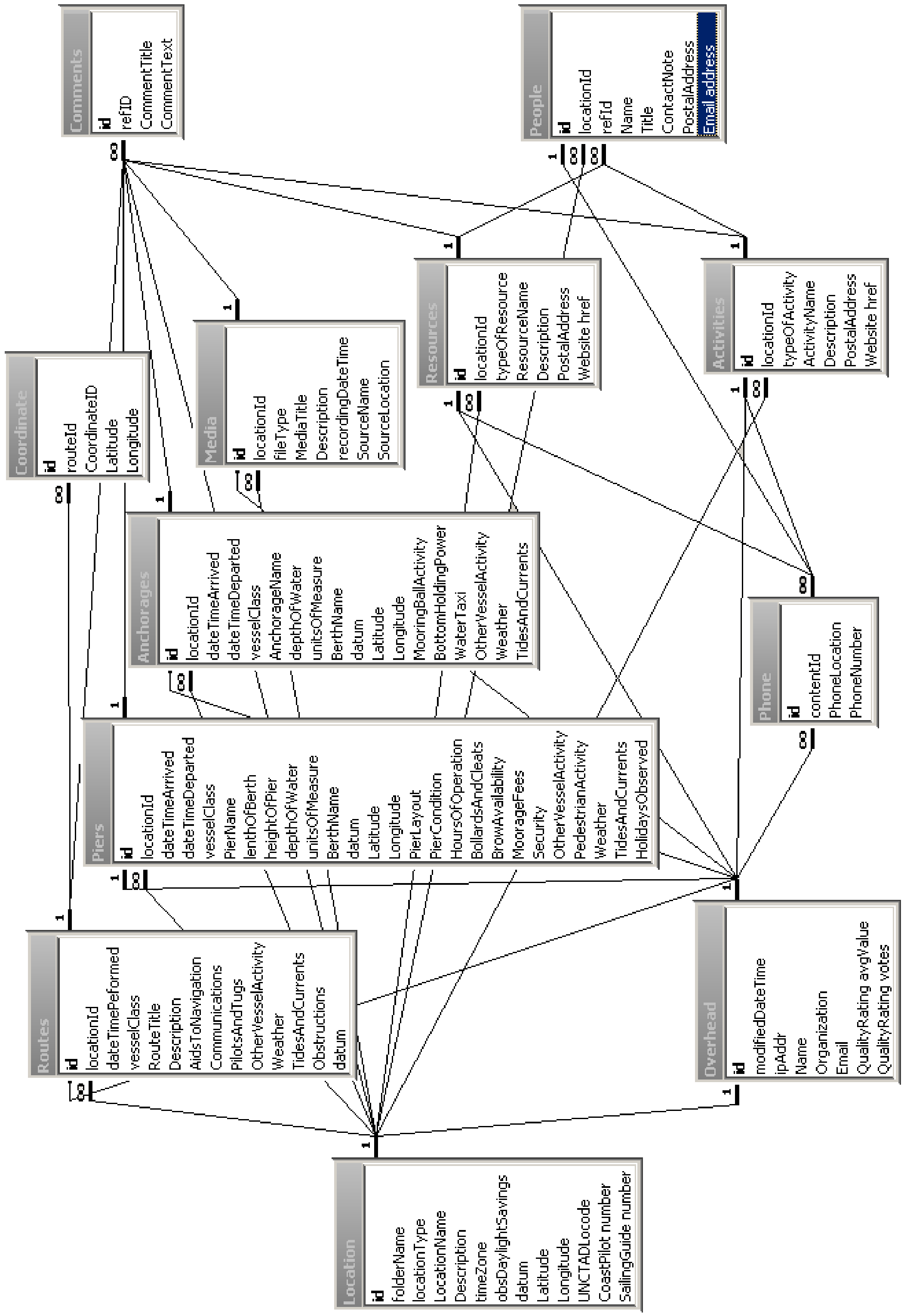
Finally, as the reliance on XML increases the importance of using namespaces, standard tag sets, reusable schemas, and non-proprietary technologies increases. Any potential implementations require research into the existing standards and organization efforts. Interoperability of XML data sources can only be possible if international, Federal, State, and local agencies develop solutions within a common definition of data types, information ontologies, and process descriptions.

C. SOURCE CODE

The source code for the application currently resides on the 'seabeeone' server ('ebiz' domain) at the Naval Postgraduate School, under the supervision of Professor Kamel. The author also maintains a copy of the same files and can be contacted at the following email address: jstewart@c2cen.uscg.mil, or via the U.S. Coast Guard locator service by calling (202) 267-0581 (or by email: locator@comdt.uscg.mil).

The appendices following this chapter provide three different views of the system. Appendix A contains an entity-relationship diagram of the data stored within the system (using Microsoft Access). The diagram does not include the parent-child relationship between locations. Appendix B contains the layout of the XML content files and their nodes. Appendix C contains sample views of data entered into the prototype application.

APPENDIX A ENTITY RELATIONSHIP DIAGRAM



THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B CONTENT FILE TEMPLATES (XML)

Location.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- A location is the primary node type within the system,
it contains generalized information about a location or area-->
<Location id="" folderName="" locationType="">

   <!-- The name of the location -->
   <LocationName></LocationName>

   <!-- A brief description about the location-->
   <Description></Description>

   <!-- Data about the time at the particular location -->
   <Time timeZone="" obsDaylightSavings=""/>

   <!-- A container for a specific waypoint -->
   <Coordinates datum="">

      <!-- Latitudes shall conform to the following regular expression:
            \d{1,2}-\d{2}(\.\d{1,4})?[NS] -->
      <Latitude/>

      <!-- Longitudes shall conform to the following regular expression:
            \d{1,3}-\d{2}(\.\d{1,4})?[EW] -->
      <Longitude/>
   </Coordinates>

   <!-- The UN Locode abbreviation for the location -->
   <UNCTADLocode/>

   <!-- The applicable Coast Pilot volume for this location -->
   <CoastPilot number=""/>

   <!-- The applicable Sailing Guide for this location -->
   <SailingGuide number=""/>

   <!-- A collection of content nodes available for this location -->
   <ContentNodes>

      <!-- A specific instance of a content node type available for this location -->
      <ContentNode contentType=""/>
   </ContentNodes>

   <!-- Overhead data included here -->
</Location>
```

Route.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Routes locationId="">
   <!-- A route has one set of comments and many waypoints -->

   <!-- A container for a specific instance of a route -->
   <Route id="" dateTimePerformed="" vesselClass="">

      <!-- The title of the route  -->
      <RouteTitle></RouteTitle>

      <!-- A container for a collection of comments -->
      <Comments>

         <!--A brief description of the route, including total length and
         number of legs, starting location and destination, etc. Also,
         important notes about the approach, entrance, and channel -->
         <Description></Description>

         <!-- Information on availability, condition, and usefulness of ATON
         present -->
         <AidsToNavigation></AidsToNavigation>

         <!-- Information on specific channels, radio stations, and comms
         procedures -->
         <Communications></Communications>

         <!-- Information on availability or use of a pilots or tugs -->
         <PilotsAndTugs></PilotsAndTugs>

         <!-- Information about the volume, types, density, and behavior
         of other vessel traffic -->
         <OtherVesselActivity></OtherVesselActivity>

         <!-- Information on unusual weather conditions en route -->
         <Weather></Weather>

         <!-- Information on unusual tides or currents observed -->
         <TidesAndCurrents></TidesAndCurrents>

         <!-- Information on hazards to navigation or obstructions present -->
         <Obstructions></Obstructions>
      </Comments>

      <!-- A container for a collection of waypoints-->
      <Waypoints datum="">

         <!-- A container for a specific waypoint -->
         <Waypoint>

            <!-- Latitudes conform to the following regular expression:
                  \d{1,2}-\d{2}(\.\d{1,4})?[NS] -->
            <Latitude/>
```

```xml
            <!-- Longitudes conform to the following regular expression:
                  \d{1,3}-\d{2}(\.\d{1,4})?[EW] -->
            <Longitude/>
        </Waypoint>
    </Waypoints>

    <!-- Overhead data included here -->
  </Route>
</Routes>
```

Piers.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Piers locationId="">
  <!-- A pier is a shoreside physical struture to moor a vessel -->

  <!-- A container for a specific instance of a pier, the title is
  contained within the primary element -->
  <Pier id="" dateTimeArrived="" dateTimeDeparted="" vesselClass="">

    <!-- Name of the pier -->
    <PierName></PierName>

    <!-- Specific data about the actual berth, and a distinct pier name -->
    <Berth lengthOfBerth="" heightOfBerth="" depthOfWater=""
unitsOfMeasure="">

      <!-- Name of the berth -->
      <BerthName></BerthName>
    </Berth>
    <!-- A container for a specific waypoint -->
    <Coordinates datum="">

      <!-- Latitudes shall conform to the following regular expression:
            \d{1,2}-\d{2}(\.\d{1,4})?[NS] -->
      <Latitude/>

      <!-- Longitudes shall conform to the following regular expression:
            \d{1,3}-\d{2}(\.\d{1,4})?[EW] -->
      <Longitude/>
    </Coordinates>

    <!-- A container for a collection of comments -->
    <Comments>
      <!-- A description of how the pier was layed out -->
      <PierLayout></PierLayout>

      <!-- Information regarding the condition of the pier, including
      construction, cleanliness, etc. -->
      <PierCondition></PierCondition>
```

```xml
      <!-- A brief summary about the hours of operation of the pier and
      its services -->
      <HoursOfOperation></HoursOfOperation>

      <!-- Information about the quanity and condition of mooring points  -->
      <BollardsAndCleats></BollardsAndCleats>

      <!-- Information regarding the availability and quality of a brow or
      gangway -->
      <BrowAvailability></BrowAvailability>

      <!-- Information moorage fees (enter office and specific person
      information in appropriate content nodes) -->
      <MoorageFees></MoorageFees>

      <!-- Information regarding the amount and quality, or lack thereof,
      of security (people, fences, gates, etc.) -->
      <Security></Security>

      <!-- Information about the types, volume, and impact of other
      vessel activity -->
      <OtherVesselActivity></OtherVesselActivity>

      <!-- Information about the volume and types of people, and
      their behavior -->
      <PedestrianActivity></PedestrianActivity>

      <!-- Information regard weather conditions observed while moored -->
      <Weather></Weather>

      <!-- Information regarding tides and currents observed specifically
      at the berth -->
      <TidesAndCurrents></TidesAndCurrents>

      <!-- Information on holidays observed and their impacts on the
      port call -->
      <HolidaysObserved></HolidaysObserved>
    </Comments>

    <!-- Overhead data included here -->
  </Pier>
</Piers>
```

Anchorages.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Anchorages locationId="">
   <!-- An anchorage is a location used to moor a vessel to the bottom with
   an anchorage or other device -->

   <!-- A container for a specific instance of an anchorage, the title is contained
   within the primary element -->
   <Anchorage id="" dateTimeArrived="" dateTimeDeparted="" vesselClass="">

      <!-- The name of the anchorage -->
      <AnchorageName></AnchorageName>

      <!-- Specific data about the actual berth, and a name if distinct from the
      general anchorage name -->
      <BerthName depthOfWater="" unitsOfMeasure="">

         <!-- The name of the berth -->
         <BerthName></BerthName>
      </BerthName>

      <!-- A container for a specific waypoint -->
      <Coordinates datum="">

         <!-- Latitudes shall conform to the following regular expression:
               \d{1,2}-\d{2}(\.\d{1,4})?[NS] -->

         <Latitude/>
         <!-- Longitudes shall conform to the following regular expression:
               \d{1,3}-\d{2}(\.\d{1,4})?[EW] -->
         <Longitude/>
      </Coordinates>

      <!-- A container for a collection of comments -->
      <Comments>

         <!-- Information regarding the availability and use of a mooring ball,
         including its condition -->
         <MooringBallAvailability></MooringBallAvailability>

         <!-- Information regarding the holding power of the bottom (separate
         from the vessel's achor used)-->
         <BottomHoldingPower></BottomHoldingPower>

         <!-- Information on availability or use of a water taxi -->
         <WaterTaxi></WaterTaxi>

         <!-- Information about the types, volume, and impact of other
         vessel activity -->
         <OtherVesselActivity></OtherVesselActivity>

         <!-- Information regard weather conditions observed while moored -->
         <Weather></Weather>
```

```xml
        <!-- Information regarding tides and currents observed specifically
        while anchored -->
        <TidesAndCurrents></TidesAndCurrents>
      </Comments>

      <!-- Overhead data included here -->
   </Anchorage>
</Anchorages>
```

Resources.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Resources locationId="">
   <!-- A resource is a entity that provides a service to the vessel
   (e.g. fuel, stores, electricity, repairs, etc.) -->

   <!-- A container for a specific instance of an resource, the name
   of the resource is contained within the primary element -->
   <Resource id="" typeOfResource="">

      <!-- The name of the resource -->
      <ResourceName></ResourceName>

      <!-- Information regarding hours of operation, rates, payment
      capabilities, quality of service, etc. about the resource -->
      <Description></Description>

      <!-- The postal address of the resource -->
      <PostalAddress></PostalAddress>

      <!-- A phone number to reach the resource -
      MORE THAN ONE Phone ELEMENT MAY EXIST -->
      <Phone location="" number=""/>

      <!-- A website address to the resource -->
      <Website href=""/>

      <!-- Overhead data included here -->
   </Resource>
</Resources>
```

Activities.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Activities locationId="">
   <!-- An activity is a service available for the morale, welfare, and
   recreation (MWR) of the crew -->

   <!-- A container for a specific instance of an activity, the name of
   the activity is contained within the primary element -->
   <Activity id="" typeOfActivity="">

      <!-- The name of the activity -->
      <ActivityName></ActivityName>

      <!-- Information regarding hours of operation, rates, payment
      capabilities, quality of service, etc. about the activity -->
      <Description></Description>

      <!-- The postal address of the activity -->
      <PostalAddress></PostalAddress>

      <!-- A phone number to reach the activity -
      MORE THAN ONE Phone ELEMENT MAY EXIST -->
      <Phone location="" number=""/>

      <!-- A website address to the activity -->
      <Website href=""/>

      <!-- Overhead data included here -->
   </Activity>
</Activities>
```

People.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<People locationId="">
   <!-- A person is an individual that may be associated with
   a resource or an activity -->

   <!-- A container for a specific instance of a person, 'refId' is used
   to associate a person to an organization (resource or activity) -->
   <Person id="" refId="">

      <!-- The name of the person -->
      <Name></Name>

      <!-- The title of the person -->
      <Title></Title>

      <!-- Information regarding this person, their duties, and any other
      relevant information -->
      <ContactNote></ContactNote>

      <!-- The postal address of the person, not necessary if same as
```

```xml
        organization's -->
        <PostalAddress></PostalAddress>

        <!-- A phone number to reach the author -
        MORE THAN ONE Phone ELEMENT MAY EXIST -->
        <Phone location="" number=""/>

        <!-- An email address for the author -
        MORE THAN ONE Email ELEMENT MAY EXIST -->
        <Email href=""/>

        <!-- Overhead data included here -->
    </Person>
</People>
```

Media.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Media locationId="">
    <!-- A media file is a non-text graphic, audio, video, slide-show, etc. -->

    <!-- A container for a specific instance of a media file.  Note that the
        name of the stored media file will be in the 'media' folder and have
        the title of the MediaFile's id value -->
    <MediaFile id="" fileType="">

        <!-- Descriptive title for the media and its content (i.e. what is it) -->
        <MediaTitle></MediaTitle>

        <!-- Elaborated comments on the content of the file -->
        <Description></Description>

        <!-- Source of the file -->
        <Source recordingDateTime="">

            <!-- The author of the file (i.e. who did the recording) -->
            <Name></Name>

            <!-- The location of creation (i.e. where was it recorded) -->
            <Location></Location>
        </Source>

        <!-- Overhead data included here -->
    </MediaFile>
</Media>
```

Comments.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Comments locationId="">
   <!-- A comment is an annotation to any other existing content -->

   <!-- The comment has its own ID and a reference to the content ID -->
   <Comment id="" refId="">

      <!-- The title of the comment -->
      <CommentTitle></CommentTitle>

      <!-- Contains feedback, clarification, addition to existing content -->
      <CommentText></CommentText>

      <!-- Overhead data included here -->
   </Comment>
</Comments>
```

Overhead.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Overhead modifiedDateTime="">
   <!-- Overhead contains information about the author and
   feedback about the quality or the content -->

   <!-- Author conatins point of contact information -->
   <Author ipAddr="">

      <!-- The name of the author -->
      <Name></Name>

      <!-- The name of the organization or unit where the author works -->
      <Organization></Organization>

      <!-- A phone number to reach the author -
      MORE THAN ONE Phone ELEMENT MAY EXIST -->
      <Phone location="" number=""/>

      <!-- An email address for the author -
      MORE THAN ONE Email ELEMENT MAY EXIST -->
      <Email href=""/>
   </Author>

   <!-- A quality score that other users provide about the content
   the author created -->
   <QualityRating avgValue="0.0" votes="0"/>
</Overhead>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDEX C SAMPLE APPLICATION VIEWS

The main Location.xml data view (XML to HTML via XSLT):

Main Location Index:

Search the system:

enter location name here [Search]

Browse the system:

- [-] 1 - United States and Canada (NIMA Region)
  - [+] 1 - Eastport to Cape Code (Coast Pilot)
  - [+] 2 - Cape Cod to Sandy Hook (Coast Pilot)
  - [+] 3 - Sandy Hook to CapeHenry (Coast Pilot)
  - [+] 4 - Cape Henry to Key West (Coast Pilot)
  - [+] 5 - Gulf of Mexico, Puerto Rico and Virgin Islands (Coast Pilot)
  - [-] 6 - California, Oregon, Washington, Hawaii (Coast Pilot)
    - [-] California (State)
      - [+] Northern California (Ocean)
      - [+] Southern California (Ocean)
    - [+] Hawaii (State)
    - [+] Oregon (State)
    - [+] Washington (State)
  - 7 - Alaska: Dixon Entrance to Cape Spencer (Coast Pilot)
  - [+] 8 - Alaska: Cape Spencer to Beaufort Sea (Coast Pilot)
  - [+] 9 - Great Lakes and Connecting Waterways (Coast Pilot)
  - [+] Canada (Country)
  - U.S. Rivers and Inland Waterways (Region)
- [+] 2 - Central and South America and Antarctica (NIMA Region)
- 3 - Western Europe, Iceland, Greenland and the Arctic (NIMA Region)
- 4 - Scandinavia, Baltic and Russia (NIMA Region)
- 5 - Western Africa and the Mediterranean (NIMA Region)

The Location.xml file for Montery, California:

Port: Monterey

Parent locations: Planet Earth > 1 - United States and Canada > 6 - California, Oregon, Washington, Hawaii > California > Southern California >

Child locations:

Description:

Summary data:

Coordinates:36-37N, 121-53W
UN Locode: US MRY
Coast Pilot: 07

Content available to view, change and add:

- Activities - Morale, welfare, and recreation
- Anchorages - Locations to anchor
- Media - Images, pictures, drawings, etc.
- People - Points of contact
- Piers - Locations to moor
- Resources - Supplies, services, etc.
- Routes - Waypoints and voyage information

New content you can add:

Other nearby locations: LONG BEACH, LOS ANGELES, SAN DIEGO, AVALON, BENI SAF, CARPINTERIA, DAVENPORT, EL SEGUNDO, ELLWOOD, GAVIOTA, ISTHMUS COVE, MORRO BAY, MOSS LANDING, NEWPORT BEACH, PISMO BEACH, PORT HUENEME, PORT SAN LUIS, PRISONERS, SANTA BARBARA, SANTA CRUZ, VENTURA, WILSON COVE,

Modify this location's data - Add a location to this area - Delete this location

Expanded activities view including navigation menu:

Activities of Monterey

Add a new activity

Local links:

- Monterey
  - Activities
  - Anchorages
  - Media
  - People
  - Piers
  - Resources
  - Routes

Type	Name	Phone	Details
Golf	Pacific Grove Golf Course	(831) 648-5775	view details
Gym/athletics	Monterey Sports Center	(831) 838-3859	hide details

Description: MSC is located across the street from the Mun Wharf No.2. The daily visit is $6. They offer huge weight rooms, a pool, a big gym, and even ping pong!

Quality rating:
0.0 avg, 0 votes.
(add your vote)

Phone numbers:

- Business - (831) 838-3859

Personnel:

(add a person to this resource)

Author: Jeff Stewart (view details)

Comments:

(add a comment) (delete all comments for this activity)

(modify this activity) (delete this activity)

Type	Name	Phone	Details
Shopping	Dell Monte Center		view details

A route view with author details shown:

Route: Monterey Bay to CG Pier at Monterey

Peformed: by a WLB, on 2002-10-17, at 07:00:00

Quality rating:
5.0 avg, 1 votes.
(add your vote)

Description of route: This fictitious route is from the center of Monterey Bay to the CG pier within Monterey harbor.

Aids to Navigation (ATON): Upon approach to the breakwall you will see a line of small green foam buoys to your starboard. Keep these buoys to port on the approach to the pier.

Communications: Station Monterey listens to channel 16 and 21. Call them at least 20 minutes before mooring (their building is about two blocks from the pier).

Other vessel activity: There are numerous small craft, tour and fishing charters, and kayaks near and within Monterey harbor. The charter boat operators listen to channel 12.

Weather: We expirienced 6 ft swells from the northwest on the approach, which quickly subsided within the breakwall and were 1.5ft at the pier. Winds NW 15kts, vis 10nm.

Tides and currents: No current within the harbor.

Obstructions: There are numerous (if not hundreds) of mooring balls within the harbor. We recommend you perform any twisting close to the pier to prevent being blown into the small craft moorings (and the small ATON).

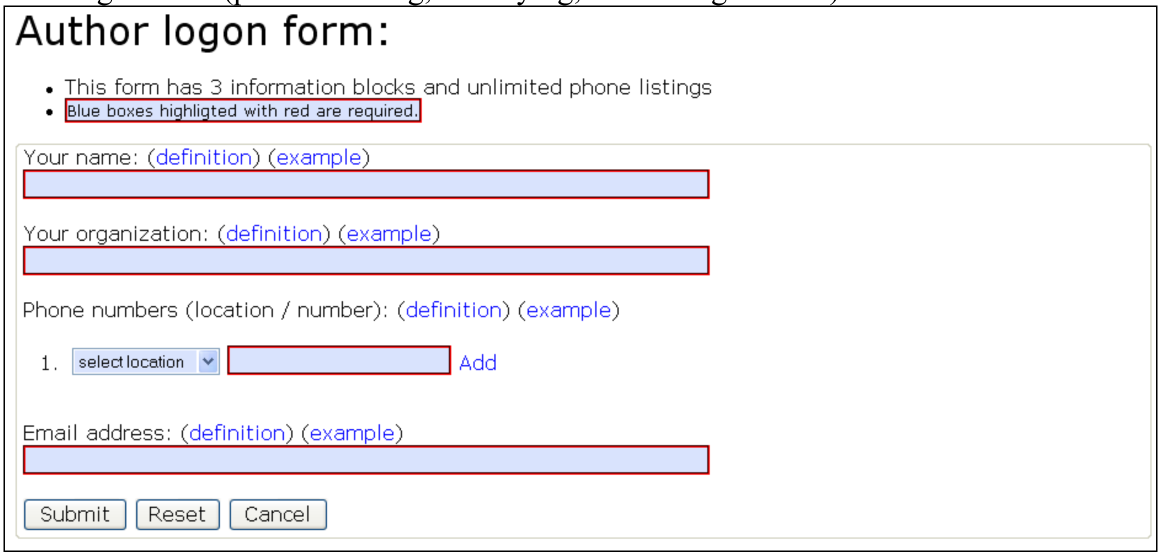
Waypoints: (WGS 84 (NAD 83) datum)

1. 36-43.10N, 122-01.12W
2. 36-37.21N, 121-53.02W
3. 36-37.19N, 121-53.03W

Author: Jeff Stewart (hide details)

Organization:	Naval Postgraduate School
Email address:	jdstewar@nps.navy.mil
Phone:	123-4567 (Work)

User logon form (prior to adding, modifying, or deleting content):



Author logon form:

- This form has 3 information blocks and unlimited phone listings
- Blue boxes highligted with red are required.

Your name: (definition) (example)

Your organization: (definition) (example)

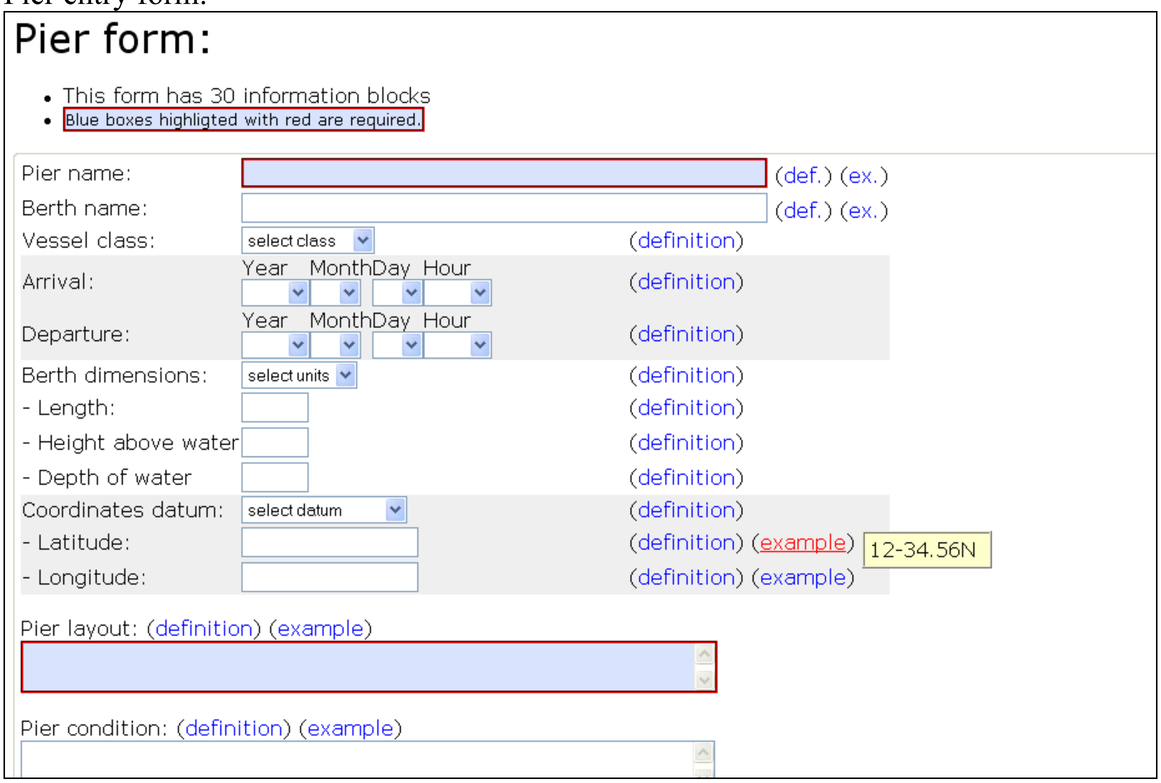
Phone numbers (location / number): (definition) (example)

1. select location Add

Email address: (definition) (example)

Submit Reset Cancel

Pier entry form:



Pier form:

- This form has 30 information blocks
- Blue boxes highligted with red are required.

Pier name:		(def.) (ex.)
Berth name:		(def.) (ex.)
Vessel class:	select class	(definition)
Arrival:	Year Month Day Hour	(definition)
Departure:	Year Month Day Hour	(definition)
Berth dimensions:	select units	(definition)
- Length:		(definition)
- Height above water		(definition)
- Depth of water		(definition)
Coordinates datum:	select datum	(definition)
- Latitude:		(definition) (example) 12-34.56N
- Longitude:		(definition) (example)

Pier layout: (definition) (example)

Pier condition: (definition) (example)

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. *Associating Style Sheets with XML Documents*. World Wide Web Consortium (W3C), 29 June 1999 <http://www.w3.org/TR/xml-stylesheet/>
2. Bourret, R.P. *XML Database Products*. 14 February 2003 <http://www.rpbourret.com/xml/XMLDatabaseProds.htm>
3. Bourret, Ronald. *XML and Databases*. 5 February 2003 <http://www.rpbourret.com/xml/XMLAndDatabases.htm>
4. Chamberlin, D. "XQuery: An XML Query Lanuage." *IBM Systems Journal*. Volume 41, Number 4, (2002): Pages 597-615.
5. Christensen, Clayton M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston: Harvard Business School Press, June 1997.
6. Comer, Douglas E. *Computer Networks and Internets, with Internet Applications*. New Jersey: Prentice Hall, 2001.
7. *Common Gateway Interface (CGI)*. National Center of Supercomputing Applications (NCSA). 4 November 2002 <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>
8. CSS Home Page. Wide Web Consortium (W3C), 13 November 2003 <http://www.w3.org/Style/CSS/>
9. *Document Object Model (DOM) Level 1 Specification*. World Wide Web Consortium (W3C), 1 October 1998 <http://www.w3.org/TR/REC-DOM-Level-1/>
10. *Document Object Model (DOM) Level 3 Load and Save Specification - W3C Working Draft*. World Wide Web Consortium (W3C), 25 July 2002 <http://www.w3.org/TR/2002/WD-DOM-Level-3-LS-20020725/>
11. *Document Object Model (DOM) Level 3 XPath Specification - W3C Working Draft*. World Wide Web Consortium (W3C), 28 March 2002 <http://www.w3.org/TR/2002/WD-DOM-Level-3-XPath-20020328/>
12. *Document Object Model Activity Statement*. World Wide Web Consortium (W3C), 2 February 2003 <http://www.w3.org/DOM/Activity>
13. *ECMA-262*. European Computing and Manufacturing Association. 13 February 2003 <http://www.ecma-international.org/publications/standards/ECMA-262.HTM>
14. *Extensible Markup Language (XML) 1.0 (Second Edition)*. World Wide Web Consortium (W3C), 6 October 2000 <http://www.w3.org/TR/REC-xml>
15. *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C), 24 February 2003, <http://www.w3.org/XML/>
16. *Extensible Stylesheet Language (XSL).* World Wide Web Consortium (W3C), 15 October 2001 <http://www.w3.org/TR/xsl/>
17. Fernandez, Mary, Paul Cotton. *XPath-XQuery Review*. World Wide Web Consortium (W3C). 28 January 2003 <http://www.w3.org/2002/Talks/www2002-xpath-xquery/>
18. Fowler, Martin, Kedall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition).* Addison-Wesley Publishing Company, 1999.
19. Henry, K. *History of CGI*. 5 November 2002 <http://206.208.128.3/khenry/tcom621/historyA.htm>
20. *HTML 4.01 Specification*. World Wide Web Consortium (W3C), 24 February 2003 <http://www.w3.org/TR/html4/>

21. Hunter, David, Kurt Cagle, Chris Dix, Roger Kovack, Jonathan Pinnock, Jeff Rafter. *Beginning XML 2nd Edition.* Birmingham: Wrox, 2001.
22. *ISO 8601 – Numeric Representations for Dates and Times*. International Organization for Standardization, 12 November 2003 <http://www.iso.ch/iso/en/prods-services/popstds/datesandtime.html?printable=true>
23. Jaworski, James. *Mastering JavaScript and JScript*. Alameda: Sybex, 1999.
24. *Locode Main*. United Nations Centre for Trade Facilitation and Electronic Business (UN/FACT), 24 February 2003 <http://www.unece.org/cefact/locode/service/main.htm>
25. Meyer, Eric C. *Cascading Style Sheets 2.0*. New York: Osborne/McGraw-Hill, 2001.
26. *Microsoft Developer Network (MSDN)*. 1 September 02 <http://msdn.microsoft.com>
27. Nambiar, U. et al. "Current Approaches to XML Managment." *IEEE Internet Computing*. July/August (2002): Pages 43-51
28. *Namespaces in XML*. World Wide Web Consortium (W3C), 14 January 1999 <http://www.w3.org/TR/REC-xml-names/>
29. *National Ocean Service – Publications & Products*. National Oceanic and Atmospheric Administration, 24 February 2003 <http://www.nos.noaa.gov/Products/products.html>
30. *NIMA Publications – World Port Index*. National Imagery and Mapping Agency, 24 February 2003 <http://pollux.nss.nima.mil/pubs/pubs_j_wpi_sections.html>
31. *SOAP Version 1.2 Part 0, 1, and 2 - W3C Candidate Recommendation*. World Wide Web Consortium (W3C), 19 December 2002 <http://www.w3.org/TR/soap12-part0/>
32. *W3C Technical Reports and Publications*. World Wide Web Consortium (W3C), 24 February 2003 <http://www.w3.org/TR/>
33. *Web Services Description Language (WSDL), Version 1.2 - W3C Working Draft*. World Wide Web Consortium (W3C), 24 January 2003 <http://www.w3.org/TR/wsdl12/>
34. *XForms - The Next Generation of Web Forms*. World Wide Web Consortium (W3C). 12 January 2003 <http://www.w3.org/MarkUp/Forms/>
35. *XForms 1.0 - W3C Candidate Recommendation*. World Wide Web Consortium (W3C), 12 November 2002 <http://www.w3.org/TR/xforms/>
36. *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) - A Reformulation of HTML 4 in XML 1.0*. World Wide Web Consortium (W3C), 1 August 2002 <http://www.w3.org/TR/xhtml1/>
37. *XML Base*. World Wide Web Consortium (W3C), 27 June 2001 <http://www.w3.org/TR/xmlbase/>
38. *XML Inclusions (XInclude) - W3C Candidate Recommendation*. World Wide Web Consortium (W3C), 17 September 2002 <http://www.w3.org/TR/xinclude/>
39. *XML Linking Language (XLink).* World Wide Web Consortium (W3C), 27 June 2001 <http://www.w3.org/TR/xlink/>
40. *XML Media Types – Request for Comments #3023*. Internet Engineering Task Force. 19 February 2003 <http://www.ietf.org/rfc/rfc3023.txt?number=3023>
41. *XML Path Language (XPath) 2.0 - W3C Working Draft*. World Wide Web Consortium (W3C), 15 November 2002 <http://www.w3.org/TR/xpath20/>

42. *XML Path Language (XPath)*. World Wide Web Consortium (W3C), 16 November 1999 <http://www.w3.org/TR/xpath>
43. *XML Schema Part 0, 1, and 2*. World Wide Web Consortium (W3C), 2 May 2001 <http://www.w3.org/TR/xmlschema-0/>
44. *XML:DB Initiative: Enterprise Technologies for XML Databases*. The XML:DB Initiative, 8 February 2003 <http://www.xmldb.org/>
45. *XPointer Framework - W3C Proposed Recommendation*. World Wide Web Consortium (W3C), 13 November 2002 <http://www.w3.org/TR/xptr-framework/>
46. *XQuery 1.0: An XML Query Language - W3C Working Draft*. World Wide Web Consortium (W3C), 15 November 2002 <http://www.w3.org/TR/xquery/>
47. *XSL Transformations (XSLT)*. World Wide Web Consortium (W3C), 16 November 1999 <http://www.w3.org/TR/xslt>
48. Larman, Craig. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*. Upper Saddle River: Prentice Hall, 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. Magdi N. Kamel
   Naval Postgraduate School
   Monterey, California

4. Prof. Gordon H. Bradley
   Naval Postgraduate School
   Monterey, California

5. CDR John Knox
   U.S. Coast Guard Headquarters (G-SRF)
   Washington, D.C.

6. Prof. Dan C. Boger
   Naval Postgraduate School
   Monterey, California